

Das Interface Age Systemhandbuch zum Commodore 64 und VC-20

Ralph Babel

Michael Krause
Andreas Dripke

DAS INTERFACE AGE SYSTEMHANDBUCH ZUM COMMODORE 64 UND VC-20

Ralph Babel

unter Mitarbeit von

Michael Krause und Andreas Dripke

Gesamtherstellung

Unternehmensberatung Andreas Dripke

Vertrieb

INTERFACE AGE
Verlagsgesellschaft

VORWORT

Die Nutzbarkeit eines Computers aus der Sicht des Programmierers hängt in entscheidender Weise von der verfügbaren Dokumentation ab. Für effiziente Programmierung ist die genaue Kenntnis der internen Strukturen unerläßlich.

In dem vorliegenden Werk wird das Betriebssystem des Commodore 64 unter Einbeziehung des VC-20 ausführlich und umfassend dargelegt. Viele der in diesem Buch enthaltenen Informationen dürften zum Zeitpunkt des Erscheinens nur wenigen Menschen auf der Erde bekannt sein. Sie finden hierin alle Daten, die für die Programmierung des Computers relevant sind.

Alle in diesem Buch genannten Fakten wurden sorgfältig recherchiert und äußerst sorgsam aufbereitet.

Die Informationen sind sowohl für den BASIC- als auch für den Assembler-Programmierer geeignet. Dabei wird es für Sie oftmals unerläßlich sein, Abschnitte mehrmals zu lesen, um sie voll zu verstehen. Die Fülle an Daten - komprimiert in einem Handbuch - macht dies erforderlich. Hätten wir einen weiter ausschweifenden Stil gewählt und damit diesem Buch mindestens dreimal so viele Seiten gegeben, so wäre es praktisch unmöglich, Daten gezielt zu suchen und insbesondere zu finden. Das Systemhandbuch ist sowohl zum Durchlesen als auch als Nachschlagewerk geeignet.

Bitte haben Sie Verständnis dafür, daß wir keine Fragen zur Programmierung beantworten können. Würden wir dies einmal anfangen, kämen wir sicherlich nicht mehr dazu, weitere Bücher für Sie zu schreiben. Wir freuen uns aber über Kritik und Anregungen von Ihnen und werden diese in einer weiteren Auflage des Werkes berücksichtigen. Und wenn Ihnen unser Buch gefällt, freuen wir uns natürlich über Ihre Empfehlung an Kollegen, Bekannte und Freunde.

Nun wollen wir Sie der faszinierenden Fülle an Daten überlassen, die Ihnen dieses Buch zu bieten hat. Möge es Ihnen ein hilfreiches Handbuch bei der Programmierung Ihres Computers sein.

Ralph Babel
Michael Krause
Andreas Dripke

Babel, Ralph; Krause, Michael; Dripke, Andreas
DAS INTERFACE AGE SYSTEMHANDBUCH ZUM COMMODORE 64 UND VC-20

1. (1983)
2. (1983)

ISBN 3-88986-001-X

Made in West Germany

Gesamtherstellung: Unternehmensberatung Andreas Dripke

Distribution(BRD): INTERFACE AGE VERLAG GMBH, Vohburgerstr.1
D-8000 München 21, Tel. (089) 5 80 67 02.

Director Sales/Marketing: Issi Franckh

International Distribution: COMPUTER BOOKBASE
(DATA DYNAMICS TECHNOLOGY, USA)
Div. of INTERFACE AGE Inc.
16704 Marquardt Avenue
Cerritos, CA 90701, U.S.A.

Druck: Offsetdruck Georg Kapehl, Wiesbaden

INTERNE CODIERUNG DER BASIC-PROGRAMME

BASIC-Befehle werden, zur Einsparung von Speicherplatz und zur Beschleunigung der Programmausfuehrung, in einen bestimmten Code umgewandelt. Ausserdem wird jeder Programmzeile noch die Information ueber die Ablageposition der naechsten Programmzeile mitgegeben, die das Suchen von Programmzeilen (zum Beispiel bei Sprungbefehlen) vereinfacht und die Suchgeschwindigkeit erhoeht. Dies geschieht beim 'Abschikken' einer Programmzeile durch die RETURN-Taste. Auch Befehle, die im Direktmodus gegeben werden, werden in die sogenannten 'Tokens' umgewandelt, bleiben jedoch im BASIC-Eingabepuffer ab Adresse 512 stehen. Daraus resultiert auch, warum Befehle, die auch den Eingabepuffer benoetigen (INPUT, INPUT# und GET(#)), im Direktmodus im Normalfall nicht ausgefuehrt werden koennen.

Die Startadresse eines BASIC-Programms steht als Pointer in den Adressen 43 und 44 und kann durch ...

```
PRINT PEEK(43) + 256 * PEEK(44)
```

... abgefragt werden. Im Normalfall duerfte das Ergebnis gleich 2049 sein, da dies die vom Betriebssystem vorgesehene Startadresse fuer BASIC-Programme ist, die jedoch fuer eigene Anwendungen geaendert werden kann.

Die ersten beiden Bytes jeder Programmzeile (bei der ersten Zeile eines Programms sind das die Speicherstellen 2049 und 2050) enthalten die Startadresse der naechsten Zeile im Adressformat, wobei (wie das in fast allen Faellen der Abspeicherung von 16-Bit-Werten ist) zuerst die Adresse low und dann die Adresse high abgespeichert wird.

Die naechsten beiden Bytes stellen die Zeilennummer der Programmzeile dar, die im Normalfall im Bereich von 0 bis 63999 liegt. Durch 'POKE' ist es jedoch auch moeglich, hoehere Zeilennummern (bis 65535) zu erzeugen, was dann aber bei Sprungbefehlen und zum Beispiel bei 'INPUT' zu sicherlich unerwuenschten Effekten fuehren kann.

Danach folgt der eigentliche Programmtext, wobei alle Befehle, Funktionen und Operatoren in Tokens codiert sind. Davon unbetroffen bleiben alle Texte, die innerhalb von Anfuehrungszeichen stehen, Bemerkungen nach 'REM' sowie Daten nach einem 'DATA'-Befehl. Der Programmtext ist mit dem Nullcode abgeschlossen.

Dieser Aufbau wiederholt sich bis zur letzten Zeile, der dann zwei Nullcodes folgen, die das Programmende markieren. Der Zeiger auf den Anfang der Variablentabelle zeigt auf das erste Zeichen nach den beiden (es sind, zusammen mit dem Nullcode zur Markierung des Endes der Programmzeile, drei aufeinanderfolgende Nullcodes) Nullcodes. Dieser Zeiger steht in den Adressen 45 und 46 und ist aequivalent dem Pointer auf den Anfang des BASIC-Programms abzufragen.

Die Laenge eines Programms kann, unabhaengig von eventuell schon definierten Variablen, durch folgende Befehlsfolge abgefragt werden ...

```
PRINT PEEK(45) + 256 * PEEK(46) - PEEK(43) - 256 * PEEK(44)
```

... was zum Beispiel bei sich selbst modifizierenden Programmen oder waehrend der Entwicklung von Programmen zur Ueberwachung der Programmlaenge verwendet werden kann.

Ein Beispiel fuer die intere Codierung

Folgendes Programm wird, nach der Eingabe von 'NEW', um ein eventuell noch vorhandenes Programm zu loeschen, in den Commodore 64 eingegeben.

```
100 rem test
500 print "Tokens
999 end
```

Diese drei Zeilen werden nun durch folgende Werte dargestellt (der jeweils erste Wert einer Zeile gibt die Startadresse der jeweiligen Zeile nur zum besseren Verstaendnis an und hat mit der Abspeicherung nichts zu tun).

```
2049  12   8 100   0 143  32  84  69  83  84   0
                     REM       t   e   s   t

2060  28   8 244   1 153  32  34 212  94  75  69  78  83   0
                     PRINT     "   T   o   k   e   n   s

2076  34   8 231   3 128   0
                     END

2082   0   0
```

Betrachten wir nun die ersten beiden Bytes der ersten Programmzeile, so stellen wir fest, dass diese auf die naechste Programmzeile zeigen (12 + 256 * 8 = 2060). Diese beiden Bytes haben nun die Werte 28 und 8 und ergeben genau die Startadresse der dritten Programmzeile (28 + 256 * 8 = 2076). Das gleiche ist auch bei der letzen Programmzeile der Fall, davon abgesehen, dass nun keine Programmzeile mehr folgt, sondern auf die beiden Nullcodes zur Kennzeichnung des Endes eines Programms gezeigt wird. Diese Pointer, die jeweils auf den Pointer der naechsten Zeile zeigen, werden auch Linkpointer genannt, da sie die Verbindung zwischen den einzelnen Zeilen herstellen.

Das jeweils dritte und vierte Byte von Programmzeilen gibt die Zeilennummer an. Da die Zeilennummern im Bereich von 0 bis 63999 liegen koennen, werden sie, ebenso wie die Linkpointer, in zwei Bytes aufgespalten. So wird die Zeilennummer '999' durch die beiden Bytes 231 und 3 (231 + 256 * 3 = 999) dargestellt.

Der Programmtext selbst ist in Einzelbytes aufgebaut. Jedem Befehl ist ein Wert zugewiesen, in diesem Beispiel sind das 'REM' mit dem Code 143, 'PRINT' mit dem Code 153 sowie 'END' mit dem Code 128.

Waehrend die Leercodes direkt nach der Zeilennummer nicht abgespeichert werden, so ist dies doch bei allen Leercodes der Fall, die ansonsten eingegeben werden.

Eine vollstaendige Aufstellung der Tokens ist der Tabelle der Interpretercodes oder dem Betriebssystemlisting zu entnehmen.

Tabelle der BASIC-Befehle, Interpretercodes und Abkuerzungen

Die Befehle sind alphabetisch geordnet, da die Auflistung in Reihe der Interpretercodes aus dem ROM-Listing ersichtlich ist.

abs	aB	182	left$	leF	200	sgn	sG	180
and	aN	175	len	len	195	sin	sI	191
asc	aS	198	let	lE	136	spc(	sP	166
atn	aT	193	list	lI	155	sqr	sQ	186
chr$	cH	199	load	lO	147	step	stE	169
close	clO	160	log	log	188	stop	sT	144
clr	cL	156	mid$	mI	202	str$	stR	196
cmd	cM	157	new	new	162	sys	sY	158
cont	cO	154	next	nE	130	tab(	tA	163
cos	cos	190	not	nO	168	tan	tan	192
data	dA	131	on	on	145	then	tH	167
def	dE	150	open	oP	159	to	to	164
dim	dI	134	or	or	176	usr	uS	183
end	eN	128	peek	pE	194	val	vA	197
exp	eX	189	poke	pO	151	verify	vE	149
fn	fn	165	pos	pos	185	wait	wA	146
for	fO	129	print	?	153			
fre	fR	184	print#	pR	152	+	+	170
get	gE	161	read	rE	135	-	-	171
go	go	203	rem	rem	143	*	*	172
gosub	goS	141	restore	reS	140	/	/	173
goto	gO	137	return	reT	142	↑	↑	174
if	if	139	right$	rI	201	>	>	177
input	input	133	rnd	rN	187	=	=	178
input#	iN	132	run	rU	138	<	<	179
int	int	181	save	sA	148			

Folgende Codes finden ausserdem Verwendung:

0	Nullcode am Zeilenende und Programmende
32 bis 95	identisch mit dem Commodore-ASCII
255	Code fuer PI

Die Codes 1 bis 31, 96 bis 127 und 204 bis 254 werden in der Grundversion des Betriebssystems nicht verwendet, werden jedoch teilweise von der Standard-BASIC-Erweiterung EXBASIC LEVEL II benutzt.

BINAERARITHMETIK

Binaeres Rechnen und Umgang mit binaeren Zahlen wird beim Arbeiten mit dem Computer immer wieder benoetigt. Ganz gleich, ob es sich dabei um die hochaufloesende Graphik, die Definition von Zeichen oder das Aendern einzelner Bits in bestimmten Registern handelt.

Eine Zahl im fuer uns gebraeuchlichen Dezimalsystem kann aus mehreren Ziffern bestehen. Dabei wird die hintere Ziffer als Einerstelle bezeichnet. Die vorletzte Ziffer ist die Zehnerstelle, die drittletzte Stelle bestimmt die Anzahl an Hundertern und so weiter. Von der Zahl eins fuer die letzte Stelle ausgehend wird der Stellenwert einer Ziffer fuer jede Stelle weiter links mit zehn multipliziert. Klar, denn unser Zahlensystem kennt schliesslich auch zehn Ziffern. Die Zahl 1983 heisst also nicht anderes als ...

1983 = 1 * 1000 + 9 * 100 + 8 * 10 + 3 * 1

Da das Binaersystem jedoch nur zwei Zustaende kennt (Strom fliesst, Strom fliesst nicht), stehen hier also auch nur zwei verschiedene Ziffern zur Verfuegung. Die Stellenwerte einzelner Ziffern koennen also nicht das zehnfache der rechts davon stehenden Ziffer betragen, sondern nur das Doppelte. Somit ergeben sich zum Beispiel fuer das Binaersystem folgende Stellenwerte: fuer die letzte Stelle, wie in jedem Zahlensystem, den Stellenwert eins, dann der Wert zwei, darauf vier, als Stellenwert fuer die viertletzte Stelle den Wert acht und entsprechend jeweils das Doppelte fuer weitere Stellen.

Da als Ziffern nur die Zahlen 0 und 1 zur Verfuegung stehen, hat eine Stelle demnach entweder den Wert null (falls die Ziffer null an dieser Stelle steht) oder den Stellenwert der entsprechenden Stelle (falls die Ziffer eins an dieser Stelle steht). Dieses eigentlich recht einfache Prinzip ist Grundlage jeglicher Operationen des Computers. Auch das Rechnen erfolgt im Binaersystem, die eingegebenen Dezimalzahlen werden fuer interne Berechnungen naemlich erst in das Binaersystem umgewandelt.

Die einzelnen Ziffern einer Binaerzahl, "Bits" genannt, werden von ihrer Position von rechts her mit Null beginnend durchnummeriert. Bit 4 zum Beispiel ist demnach die fuenfte Stelle einer Binaerzahl (von rechts!). Die Stellenwerte lassen sich durch diese Vereinbarung sehr leicht errechnen. Es muss nur die Bitposition zur Basis zwei potenziert werden. Fuer Bit 4 ergibt dies einen Stellenwert von 2 ↑ 4 (lies: zwei hoch vier), entsprechend dem Dezimalwert 16. Ein Bit (oder eine bestimmte Bitposition) zu SETZEN heisst nun nichts anderes, als dieser Binaerstelle den Ziffernwert '1' zuzuweisen. Das LOESCHEN eines Bits entspricht demnach der Zuweisung des Ziffernwerts '0' an eine Bitposition.

Da der Commodore 64 einen Prozessor mit einem "8-Bit-Datenbus" besitzt, werden jegliche Speicheroperationen immer mit acht Bits gleichzeitig ausgefuehrt. Werden Daten aus dem Speicher gelesen, so werden immer acht Bits gelesen, so auch bei den Befehlen "POKE" und "PEEK", die immer Daten mit einer Breite von acht Bits schreiben beziehungsweise lesen. Bei der sogenannten "vorzeichenlosen Binaerarithmetik" mit acht Bits haben die einzelnen Stellen folgende Stellenwerte:

Bitposition	7	6	5	4	3	2	1	0
Stellenwert	128	64	32	16	8	4	2	1

Der Minimalwert (alle Bits geloescht), den eine vorzeichenlose Binaerzahl annehmen kann, ist demnach der Wert 0, der Maximalwert (alle Bits gesetzt) einer achtstelligen Binaerzahl ist 255. Eine solche Binaerzahl mit einer Breite von acht Bits nennt man auch "Byte". Ein Byte ist also gleichzusetzen mit acht Bits.

Zusaetzlich zu den Bytes existieren nun noch die "Adressen". Eine Adresse ist eine vorzeichenlose Binaerzahl, die aus sechzehn Bits (entsprechend zwei Bytes) besteht. Durch eine Adresse kann die Position bestimmter Stellen im Arbeitsspeicher festgelegt werden. Zwei aufeinanderfolgende Bytes im Arbeitsspeicher, die auf eine andere Stelle verweisen, nennt man "Pointer".

Hierzu gleich zwei Programme, die die Umwandlung einer natuerlichen Dezimalzahl ins Binaersystem und umgekehrt vornehmen:

Binaer nach Dezimal:

```
100 INPUT B$ : D = 0 : FOR I = 1 TO LEN (B$)
110 S$ = MID$ (B$, I, 1)
120 IF S$ < "0" OR S$ > "1" THEN I = LEN (B$) : NEXT : END
130 D = D * 2 + ASC (S$) - 48 : NEXT : PRINT D : END
```

Dezimal nach Binaer:

```
200 INPUT D : B$ = ""
210 R = INT (D / 2) : B$ = CHR$ (D - R * 2 + 48) + B$
220 D = R : IF R GOTO 210
230 PRINT B$ : END
```

Bei der Umwandlung ins Binaersystem werden die Binaerzahlen so umgewandelt, dass fuehrende Nullen unbeachtet bleiben. Ist eine Darstellung des Binaeraequivalents von Dezimalzahlen im Bereich von 0 bis 255 in Form von acht Binaerstellen erwuenscht (Auffuellen auf eine Laenge von acht Ziffern), so ist folgende Programmzeile hinzuzufuegen:

```
225 IF LEN (B$) < 8 THEN B$ = RIGHT$ ("0000000" + B$, 8)
```

Allerdings kann ein Byte nicht nur die Funktion der Darstellung einer Zahl haben (arithmetische Funktion). Es ist moeglich, dass jedem Bit in einem Byte eine bestimmte Funktion zukommt (logische Funktion). So koennen anhand eines Bytes insgesamt acht Bedingungen mit je zwei Moeglichkeiten abgeprueft werden. Man nennt in diesem Fall ein solches Bit, mit der Bedeutung als Hinweis fuer eine Ja/Nein-Entscheidung, eine "Flag", was soviel heisst wie "Flagge".

Solche "Flags" treten besonders in bestimmten "Registern" auf. Ein Register ist ein Byte innerhalb des Arbeitsbereichs des Computers, das eine bestimmte Funktion erfuellt. Dies sind zum Beispiel die Register des VIC-II-Chips, anhand derer bestimmte Funktionen des Video-Chips festgelegt werden, oder viele Adressen im Bereich von 0 bis 1023.

Zur Manipulation von Binaerzahlen stehen beim Commodore-BASIC zwei Befehle zur Verknuepfung zweier Binaerzahlen zur Verfuegung: "AND" und "OR". Die beiden Operatoren bewirken, dass die beiden Argumente als Binaerzahlen Bit fuer Bit (insgesamt verarbeiten die Befehle 16 Bits, jedoch mit Vorzeichen, was aber beim Arbeiten mit 8-Bit-Zahlen unbeachtet bleiben kann) verglichen werden und gleichzeitig das Ergebnis aufgebaut wird. Werden zwei Zahlen mittels "OR" verknuepft, so sind in der Ergebnis-Binaerzahl saemtliche Bitpositionen geloscht, bei denen beide Bitpositionen der beiden Argumente geloescht waren, alle anderen Bits sind gesetzt. Bei "AND" sind entsprechend alle Bits gesetzt, bei denen beide Bitpositionen der Argumente gesetzt waren, andere Bitpositionen sind geloescht. Dadurch laesst sich nun bewirken, dass innerhalb eines Registers durch "PEEK" und "POKE" einzelne Bits gesetzt und geloescht werden koennen.

Soll zum Beispiel ein bestimmtes Bit (mit dem Stellenwert N) an der Adresse A gesetzt werden, so kann dies durch ...

```
POKE A, PEEK (A) OR N
```

... erfolgen. Dies erklaert sich daraus, dass jegliche Bits, die durch "OR" mit dem Ziffernwert '0' verknuepft werden, unbeeinflusst bleiben. Wird jedoch eine Verknuepfung durchgefuehrt, bei der mindestens eins der beiden Bits gesetzt ist, so ist auch im Ergebnis dieses Bit gesetzt.

Aehnlich verlaeuft dies beim Loeschen von Bits, was aber nur mittels "AND" moeglich ist. Wird naemlich ein Bit durch "AND" mit dem Wert '0' verknuepft, so ist das Ergebnisbit auch geloescht. Allerdings muss nun erreicht werden, dass die uebrigen Bits unbeeinflusst bleiben. Dies ist dann der Fall, wenn sie bei der "AND"-Verknuepfung mit dem Wert '1' verknuepft werden. Es muss also nun mit einem Byte gearbeitet werden, bei dem das Bit, das geloescht werden soll, den Bitwert '0' hat, alle uebrigen Bits den Wert '1'. Entsprechend dem Beispiel fuer das Setzen von Bits erreicht man das Loeschen durch ...

```
POKE A, PEEK (A) AND (255 - N)
```

Die Klammern fuer die Subtraktion dienen nur der Uebersicht und koennen entfallen, da der Operator '-' einer hoehere Prioritaet als 'AND' hat und daher auf jeden Fall vorher ausgefuehrt wird.

Sollen mehrere Bits gleichzeitig entweder gesetzt oder geloescht werden, so kann dies durch Addition der Stellenwerte erfolgen.

Zur Bezeichnung von Binaerziffern und zusammengesetzten Binaerziffern existieren noch weitere gebraeuchliche Begriffe: So zum Beispiel MSB und LSB (Most/Least Significant Byte): Ein MSB ist das Byte innerhalb einer Kette von Bytes, dem der hoechste Stellenwert zukommt. Bei einer Adresse waere dies das Byte mit den Stellenwerten von 256 bis 32768. Das LSB ist entsprechend das niederwertigste Byte. Bei Adressen sind nur zwei Bytes verwendet, jedoch findet sich dieser Begriff auch bei der Verwendung von Binaerziffern mit mehr als 16 Stellen (Bits). Die Begriffe MSB und LSB werden jedoch auch im Zusammenhang mit Bits verwendet. Hierbei ist dann mit MSB (Most Significant Bit) das hoechstwertige Bit (innerhalb eines oder mehrerer Bytes) gemeint.

Auch gebraeuchlich sind (oft auch im Zusammenhang mit Zahlen im Hexadezimalsystem oder gepackten BCD-Ziffern) die Bezeichnung MSN und LSN, die fuer die Bezeichnung von "Nybbles" verwendet werden. Ein Nybble ist ein halbes Byte, also vier Bits. Ein Byte besteht daher aus zwei Nybbles, das eine besteht aus den Bits der Positionen 0 bis 3 (LSN), das andere entsprechend aus den Bits 4 bis 7 (MSN).

Noch eine kurze Anmerkung zum Hexadezimalsystem: Dieses Zahlensystem baut auf der Basis 16 (entsprechend 16 verschiedenen Ziffern) auf. Diese Ziffern sind die Zahlen von '0' bis '9' sowie die Buchstaben von 'A' bis 'F', wobei die Buchstaben fuer die Dezimalwerte von 10 bis 15 stehen. Dieses Zahlensystem fand und findet hauptsaechlich auf Einplatinensystemen Anwendung, wo die Moeglichkeit einer Programmierung im Dezimalsystem ein zu aufwendiges Monitorprogramm benoetigen wuerde (Hexadezimalzahlen lassen sich leichter ins fuer den Computer verstaendliche Binaersystem umsetzen). Doch mit dem Aufkommen von groesseren Computern wurde in vielen Faellen dieses fuer den Menschen doch sehr ungebraeuchliche Zahlensystem mituebernommen, anstatt dem Programmierer durch Verwendung des Dezimalsystems entgegenzukommen. Mittlerweile haben sich viele Programmierer fuer die Assemblerprogrammierung im Dezimalsystem entschlossen, da dieses ein wesentlich komfortableres Arbeiten erlaubt. Das Hexadezimalsystem findet seine Berechtigung nur auf Minimalsystemen und nicht auf Computern, die im Normalfall (BASIC, PASCAL etc.) sowieso zum Arbeiten im Dezimalsystem konzipiert sind.

Zurueck zum Binaersystem: Es kann auch vorkommen, dass Dezimalzahlen mit Nachkommastellen ins Binaersystem umgewandelt werden muessen. Hierbei wieder ein Vergleich mit dem Dezimalsystem: Die Stelle vor dem Komma (die Einerstelle) hat ein Zehntel des Wertes der Stelle links davon (Zehnerstelle), demnach hat die erste Stelle nach dem Komma den Wert '0.1', die naechste den Wert '0.01' und so weiter. Im Binaersystem hat demnach die erste Nachkommastelle den Wert '0.5' (hier wird naemlich durch zwei dividiert, nicht durch zehn), die zweite Nachkommastelle den Wert '0.25', die naechste '0.125'. Soll eine Dezimalzahl mit Nachkommastellen ins Binaersystem umgewandelt werden, so hat zuerst die Umwandlung des Vorkommateils zu erfolgen. Der Nachkommateil wird nun verdoppelt (die Binaerzahl wird also um eine Ziffer nach LINKS verschoben). Die Ziffer, die sich nun als Vorkommastelle ergibt (entweder '0' oder '1'), ist die erste Nachkommastelle. Mit dem Nachkommateil dieser verdoppelten Zahl wird nun genauso verfahren: Verdopplung und Verwendung der Vorkommastelle als naechste Nachkommastelle. Daraus resultiert natuerlich, dass nicht alle Dezimalzahlen sich abbrechend ins Binaersystem umwandeln lassen. So ist zum Beispiel die Dezimalzahl '0.1' eine periodische Binaerzahl:

0.1 (dezimal) = 0.0001100110011001100110011... (binaer)

Ausser der Darstellung von gebrochenen Zahlen wird beim Umgang mit Binaerzahlen nun noch die Darstellung von negativen Zahlen benoetigt: Eine Moeglichkeit der Darstellung von Binaerzahlen ist, das Vorzeichen in Form einer Flag zusammen mit der Zahl abzuspeichern. Diese Methode wird bei der internen Darstellung von Fliesskommazahlen beim Commodore-BASIC (siehe dort) verwendet. Es wird nur der Betrag der Zahl abgespeichert, aber zusammen mit einem Bit, das angibt, ob die Zahl positiv oder negativ ist.

Bei der gebraeuchlicheren Methode erfolgt die Darstellung in Form des sogenannten "Zweierkomplements". Erst einmal zum Begriff der Komplements: das "Einserkomplement" einer Binaerzahl ergibt sich aus der invertierten Darstellung der Ausgangszahl. Jede Ziffer wird durch die entgegengesetzte Ziffer dargestellt, die '0' wird durch die '1' ersetzt und umgekehrt. So ist das Komplement von '01101110' gleich '10010001'. Das Zweierkomplement entspricht nun dem um den Wert eins erhoehten Einserkomplement. Hierzu ein Beispiel:

12 (dezimal) = 00001100 (binaer)
Einserkomplement von 00001100 (binaer) = 11110011 (binaer)
11110011 (binaer) + 1 (binaer) = 11110100 (binaer)

Wird nun die positive Zahl zu ihrem Zweierkomplement addiert, so ist das Ergebnis gleich null.

Aus dieser Darstellung einer Binaerzahl mit Vorzeichen resultiert jedoch jetzt, dass Bit 7 (falls nur mit acht Bits gearbeitet wird) nun nicht mehr den Stellenwert 128, sondern vielmehr -128 hat. Addiert man nun die Ziffernwerte der Zahl '-12' aus dem obigen Beispiel (-128 + 64 + 32 + 16 + 4), so erhaelt man auch den korrekten Wert. Auch beim Commodore-BASIC wird bei den Operatoren "AND", "OR" und "NOT" (dieser Befehl errechnet das Einserkomplement) mit vorzeichenbehafteten Binaerzahlen gerechnet, jedoch mit Binaerzahlen einer Breite von sechzehn Bit. Daraus ergibt sich, dass Bit 15 nicht den Wert 32768, sondern -32768 hat (Bit 7 behaelt den normalen Wert von 128). Wird zum Beispiel eingegeben ...

```
PRINT NOT 0
```

... so erhaelt man den auf den ersten Blick unsinnigen Wert '-1' als Ergebnis. Da jedoch das Einserkomplement von '0' (oder in Binaer '0000 0000 0000 0000') gleich '1111 1111 1111 1111' ist, erhaelt man nun durch Addition aller Stellenwerte den Wert '-1'. Auch die Statusvariable 'ST' wird als 8-Bit-Zahl mit Vorzeichen angesehen. Dies kann man jedoch umgehen durch ...

```
PRINT 255 AND ST
```

... da hierdurch der Wert von ST (8 Bits) zuerst in 16 Bits umgewandelt wird (die Ziffer von Bit 7 wird in die Positionen 8 bis 15 uebertragen), dann werden die Bits 8 bis 15 abgeschnitten. Da jedoch bei logischen Operatoren nicht Bit 7, sondern Bit 15 gesetzt sein muss, um negative Zahlen darzustellen, wird nun der Wert von 'ST' so ausgegeben, als sei er eine vorzeichenlose Zahl. Der Wertebereich von 8-Bit-Zahlen mit Vorzeichen ist -128 bis 127, bei 16-Bit-Zahlen von -32768 bis 32767 (beim Commodore-BASIC kann die Zahl -32768 als Ergebnis einer logischen Operation auftreten, als Argument wird sie jedoch durch die Fehlermeldung "?ILLEGAL QUANTITY ERROR" zurueckgewiesen).

Eine weitere Darstellungsweise fuer Zahlen ist das gepackte BCD (binary coded decimal) Format (wird im Dezimalmodus der 65er CPUs direkt verarbeitet, andere CPUs besitzen Befehle zur Anpassung): Hier entspricht jedes Nybble eines Bytes einer Dezimalstelle. Jedoch nimmt dieses Nybble jetzt nicht mehr die Werte von 0 bis 15, sondern nur noch die Werte von 0 bis 9 an. Ein Byte, das eine Zahl im gepackten BCD-Format enthaelt, kann daher jetzt nur noch Zahlen im Bereich von 0 bis 99 enthalten. Zum Beispiel wird die Dezimalzahl '49' im BCD-Format als '0100 1001' gespeichert. Das MSN hat den Wert 4, das LSN den Wert 9.

DARSTELLUNG VON FLIESSKOMMAZAHLEN IM CBM-BASIC

Das BASIC des Commodore 64 bietet die Fliesskommazahlen als Variablentyp an. Die Darstellung derselben unterscheidet sich jedoch so stark von der Darstellung von Integervariablen (die problemlos ins Dezimalsystem rueckuebersetzbar sind), so dass hier gesondert darauf eingegangen werden soll. Fliesskommawerte werden innerhalb der Variablentabelle in fuenf Bytes abgespeichert (siehe Darstellung von Variablen). Auf die Bedeutung dieser fuenf Bytes soll im folgenden eingegangen werden.

Als Beispiel dient hier die Abspeicherung der Zahl '-9876.25'. Der erste Umwandlungsschritt ist die Darstellung dieser Zahl im Binaersystem. Das Vorzeichen bleibt hierbei unberuecksichtigt, da negative Zahlen nicht in Form des um eins erhoehten Einserkomplements dargestellt werden, sondern das Vorzeichen getrennt angegeben wird. Die Zahl '9876.25' sieht im Binaersystem nun wie folgt aus:

    9876.25 (dezimal) = 10011010010100.01 (binaer)

Nun lassen sich Binaerzahlen, wie Zahlen eines jeden anderen Zahlensystems auch, in Exponentialdarstellung bringen. So wie die Dezimalzahl ...

    25000 zu 2.5E+04

... wird, so laesst sich auch die obige Binaerzahl in Exponentialdarstellung (zur Basis zwei) bringen. Wird nun das Komma (unter Erhoehung des Exponenten) so verschoben, dass keine Zahl mehr vor diesem steht (und direkt nach dem Komma eine eins), so entspricht dies schon im ungefaehren der Darstellung im CBM-BASIC. Die Zahl im obigen Beispiel wuerde nun wie folgt aussehen:

    10011010010100.01 = 0.1001101001010001 E 1110

... saemtliche Zahlen im Binaersystem. Der Exponent entspricht der Dezimalzahl 14, da das Komma schliesslich um 14 Stellen nach links verschoben wurde. Bei einer Verschiebung nach rechts waere der Exponent negativ. Zu diesem Exponenten wird nun der Wert 128 addiert, man erhaelt das erste Byte des Variablenwerts bei Real-Variablen, aus dem man wiederum den Exponenten errechnen kann. Um nun die restlichen vier Bytes zu erhalten, muss folgendermassen vorgegangen werden: Man unterteilt die Mantisse der Binaerzahl in vier Gruppen zu je acht Ziffern (falls nicht genug Ziffern vorhanden sind, wird der Rest mit Nullen aufgefuellt) und ersetzt die erste Ziffer (das war der Ziffernwert eins, der eine Stelle rechts des Kommas stand) im ersten Block mit acht Ziffern durch die Vorzeichenflag (positiv = 0, negativ = 1). Das Komma kann ab hier entfallen, es ist nur die Ziffernfolge von Bedeutung. Hier wieder unser Beispiel ...

Mantissenziffernfolge: 1001101001010001

In Bloecke unterteilt: 10011010 01010001 00000000 00000000

Ersetzung der ersten Ziffer durch die Vorzeichenflag:

    1 0011010 01010001 00000000 00000000

In diesem Fall aendert sich die erste Ziffer nicht, da wir ja von einer negativen Zahl ausgegangen sind. Waere die Zahl positiv, so waere die erste Ziffer eine Null.

Die nun folgenden Umwandlungsschritte macht der Rechner nun nicht mehr mit, da er selbst ja normalerweise auch im Binaersystem arbeitet und so die Zahlenbloecke nicht mehr zur weiteren Verarbeitung ins Dezimalsystem konvertieren muss.

Wie gesagt werden diese vier Bloecke zu acht Ziffern (Bits) nun ins Dezimalsystem umgewandelt. Man erhaelt so die weiteren vier Bytes des Variablenwerts. Im Endeffekt wird unsere Zahl (falls sie einer Variablen zugewiesen wird) abgespeichert als ...

142 154 81 0 0

Das erste Byte ist der (Binaer-) Exponent (zur Basis 2, um 128 erhoeht), der Rest stellt die Mantisse und das Vorzeichen dar.

Die Zahlendarstellung in den Rechenregistern (FAC und ARG, auch Fliesskommaakkumulatoren #1 und #2 genannt) des Betriebssystems ist ein wenig anders. Hier wird das fuer das Vorzeichen zustaendige Bit wieder durch eine eins ersetzt und das Vorzeichen selbst in einer getrennten Zeropage-Adresse abgespeichert. Zusaetzlich besitzt FAC (Floatingpoint ACcumulator) im Unterschied zu ARG (floatingpoint ARGument) noch eine Rundungsstelle, die einem fuenften Mantissenbyte gleichkommt.

DARSTELLUNG UND ABLAGE VON VARIABLEN

Die Variablen werden im Commodore-BASIC in drei Bloecken variabel verwaltet. Es handelt sich dabei um die einfachen (nichtindizierten) Variablen, die indizierten Variablen, Felder oder Arrays genannt, sowie die Inhalte der Stringvariablen, die in einem weiteren Datenblock verwaltet werden.

Der erste Block liegt direkt hinter dem BASIC-Programm, wodurch auch klar wird, warum Variablen durch Programmaenderungen geloescht werden - es werden beim Einbau von Programmzeilen die Variablen einfach ueberschrieben. Der Pointer (45/46) zeigt auf den Beginn des Variablenbereichs, das Ende wird durch den Anfangspointer auf den Block mit den indizierten Variablen bestimmt, die direkt hinter den nichtindizierten stehen. Diese Startadresse der Feldvariablentabelle wird durch den Pointer (47/48) festgelegt. Die Endadresse (plus eins, wie alle Endpointer) steht in (49/50).

Die Tabelle der nichtindizierten Variablen ist in Eintraege zu je sieben Bytes aufgeteilt, die den Variablennamen und den Inhalt (beziehungsweise den Pointer auf den Inhalt) enthalten. Bei den Eintraegen handelt es sich um die Real-Variablen, die Integer-Variablen, die Strings sowie die mittels 'DEF' definierten "Variablen". Der Aufbau eines Eintrags in folgender Tabelle in einer Uebersicht:

Typ	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
real	reset	reset	Expon.	Variablenwert von MSB bis LSB			
$	reset	set	Laenge	Startadresse LH		0	0
FN	set	reset	Startadresse LH		Variablenp. LH		-
%	set	set	MSB	LSB	0	0	0

Der Variablenname steht in den ersten beiden Bytes, und da Zeichen des Variablennamens nur Buchstaben oder Zahlen sein duerfen, wird Bit 7 zusaetzlich zur Markierung des Variablentyps verwendet. Die Angabe 'set' beziehungsweise 'reset' bezieht sich also auf Bit 7 des Variablennamens. Die weiteren fuenf Bytes geben, sofern benutzt, den Variablenwert an. Die Aufteilung dieser fuenf Bytes bei den Reals und Integers bedarf keiner weiteren Erklaerung.

Die Angabe 'Startadresse LH' bei Eintraegen von Stringvariablen bezieht sich auf die Startadresse (erst low, dann high) des Stringinhalts, der entweder im Programm (bei Direktzuweisungen) oder am Ende des Arbeitsspeichers liegen kann (Stringoperationen oder Eingaben). Ist die Stringlaenge gleich null, so ist die Startadresse natuerlich egal.

FN-Variablen enthalten einerseits den Pointer auf den Ausdruck, der auszuwerten ist, sowie den Pointer auf die FN-Variable, die im Ausdruck verwendet wird. Beim Aufruf einer selbstdefinierten Funktion wird der Inhalt der FN-Variablen gerettet, das Argument des FN-Ausdrucks in die Variable uebertragen, der Ausdruck ausgewertet (bei Fehlern im Ausdruck wird die Zeilennummer, in der das FN steht, ausgegeben, nicht die des FN-Ausdrucks) und zur weiteren Verwendung in FAC uebertragen. Danach wird der Inhalt der FN-Variablen wiederhergestellt und das Programm an der Stelle hinter dem FN-Aufruf fortgesetzt.

Das Suchen und Anlegen von Variablen kann durch die Routine 'VARSUC' (ab 45195) durchgefuehrt werden.

Der Block am Ende des Arbeitsspeichers enthaelt die Stringinhalte, sofern sie nicht direkt im Programm zugewiesen wurden. Diese Stringinhalte reichen bis zum Ende des Arbeitsspeichers (Endadresse im Pointer (55/56)) und werden nach unten hin angebaut. Saemtliche Zwischenergebnisse bei Stringoperationen werden abgespeichert und dann zu weiteren Berechnungen herangezogen. Ist ein Ausdruck ausgewertet, so werden die "Reste", die Zwischenergebnisse und frueheren Stringinhalte, nicht beseitigt, sondern unveraendert im Block gelassen, dessen Anfang, festgelegt durch den Pointer (51/52), sich immer weiter in Richtung der Variablentabelle bewegt. Wird nun beim Einbau einer neuen Variablen, eines Variablenfeldes oder eines weiteren Strings mehr Platz benoetigt, als momentan verfuegbar ist, so werden diese ungueltigen Strings durch eine Routine namens "GARBAGE COLLECT" (woertlich uebersetzt: Abfallsammlung) beseitigt. Ist nun immer noch nicht genug Platz vorhanden, so wird ein "OUT OF MEMORY" ausgegeben, ansonsten wird die Ausfuehrung normal fortgesetzt. GARBAGE COLLECT kann durch die Funktion 'FRE(0)' erzwungen werden.

Man sollte uebrigens, wenn ein Programm grosse Felder verwendet, aus Zeitgruenden normale Variablen VOR der Dimensionierung von Feldern anlegen, da sonst, um eine Variable einzufuegen, der gesamte Block mit Feldvariablen durch eine sehr zeitaufwendige Routine um sieben Bytes nach oben verschoben werden muss.

Nun zum Aufbau von Feldern: Eintraege einzelner Elemente von Feldvariablen sind, im Gegensatz zu nichtindizierten Variablen, unterschiedlich lang. So benoetigt eine Realvariable zwar, wie sonst auch, fuenf Bytes, um den Wert darzustellen, Integer- und Stringvariablen jedoch werden nun nicht mehr mit Nullen aufgefuellt, sondern nur noch so lang abgespeichert, wie Werte verwendet werden; zwei Bytes bei Integervariablen und drei Bytes bei Strings.

Zu Beginn eines Feldes steht der sogenannte Arrayheader, ein Kopfeintrag, der die Informationen ueber das Feld enthaelt. Dies sind Angaben ueber die Anzahl Dimensionen, die Spaltenlaenge(n), die Feldlaenge und natuerlich der Name des Feldes. Auch hier sind die Namensmarkierungen identisch denen bei nichtindizierten Variablen, auch wenn es natuerlich keine Felder vom Typ 'FN' gibt. Ist das zweite Byte eines Variablennamens nicht benutzt, so wird auch hier der Code 0 beziehungsweise 128 (abhaengig vom Variablentyp) als zweites Zeichen gespeichert.

Hier ein schematischer Aufbau des Arrayheaders:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
erstes Zeichen	zweites Zeichen	Feldlaenge in LH Darstellung		Anzahl Dimens.	Spaltenlaenge HL Darstellung!	

Byte 5 und 6 werden, je nach Anzahl an Dimensionen, wiederholt. Dabei muss beachtet werden, dass die letzte Spaltenangabe beim DIM-Befehl als erster Eintrag abgespeichert wird. Ausserdem wird die Spaltenlaenge und nicht die Zahl beim DIM-Befehl im Arrayheader eingetragen, da schliesslich noch ein Element mit dem Index Null existiert (der Wert im Kopfeintrag ist also um eins hoeher als der Parameter im DIM-Befehl).

Hierzu ein Beispiel: ein Stringfeld soll durch folgenden DIM-Befehl angelegt werden:

```
DIM X5$ (2,5,567)
```

Das Feld wird nun folgendermassen im Arbeitsspeicher abgelegt:

Feldadresse plus ...	Inhalt	Bedeutung
0	88	Code fuer "X" (Bit 7 geloescht)
1	181	Code fuer "5" (Bit 7 gesetzt)
2	219	Feldlaenge in Low/High Darstellung
3	119	(ist hier gleich 30683)
4	3	Anzahl an Dimensionen des Feldes
5	2	Wert 568 fuer Spaltenlaenge
6	56	(High/Low Darstellung!)
7	0	Wert 6 fuer Spaltenlaenge
8	6	
9	0	Wert 3 fuer Spaltenlaenge
10	3	
11	-	Stringlaenge fuer Element (0,0,0)
12	-	zugehoerige Startadresse in
13	-	Low/High Darstellung
...	...	10222 weitere Stringelemente
30680	-	Stringlaenge fuer Element (2,5,567)
30681	-	zugehoerige Startadresse in
30682	-	Low/High Darstellung

Die Zaehlweise der Elemente innerhalb des Feldes geht folgendermassen vor sich: Die erste Spalte ist die "niederwertigste Spalte", die letzte die "hoechstwertigste". Der auf den ersten Eintrag im obigen Beispiel folgende Eintrag waere der des Elements (1,0,0), darauf (2,0,0) und dann (0,1,0) und so weiter.

Nun ist es auch einfach, eine Methode zur Berechnung der Laenge eines Feldes zu entwickeln. Man multipliziert die um eins erhoehten (Null-Eintrag!) Parameter des DIM-Befehls miteinander, multipliziert dies mit der Laenge eines Eintrags (Real = 5, String = 3, Integer = 2) und addiert dazu den Wert fuenf und die doppelte Anzahl Dimensionen. Das obige Feld belegt nach dieser Formel ...

```
(2+1) * (5+1) * (567+1) * 3 + 5 + 3 * 2
```

... gleich 30683 Bytes. Dies ermoeglicht es in Programmen, den Anwender auf eine freundlichere Art als "OUT OF MEMORY" darauf aufmerksam zu machen, dass der Arbeitsspeicher nicht ausreicht. Man muss vor Ausfuehrung des DIMs nur die Feldlaenge mit dem noch freien Platz vergleichen.

DIE VERWENDUNG DER USR-FUNKTION

Das Commodore-BASIC verfuegt ueber eine Funktion mit dem Namen "USR". Waehrend man im Normalfall durch Verwendung dieser Funktion nur einen "?ILLEGAL QUANTITY ERROR" erzielen kann, so koennen durch Maschinenprogramme der "USR"-Funktion viele Bedeutungen (fast) aller Art zugewiesen werden. Eine dieser speziellen Aufgaben ist zum Beispiel die "PEEK"-Funktion, die immer den Inhalt des RAMs liest (siehe Speicherverwaltung). Auch das Arbeiten mit der internen Uhr (siehe CIA, Time of Day) erfaehrt eine erhebliche Vereinfachung, wenn man ein geeignetes Programm zur Bedienung der Uhr anwendet. Um ein problemloses Implementieren eigener Funktionen so einfach wie moeglich zu machen, hier eine Beschreibung:

Die Funktion "USR" ist eine Funktion wie jede andere auch, sei es nun eine mathematische Funktion wie "SIN" oder "EXP", eine spezielle Funktion im Zusammenhang mit Strings (zum Beispiel "VAL" oder "CHR$") oder nur etwas systemspezifisches wie "PEEK" oder "FRE". Allen diesen Funktionen, deren Interpretercodes im Bereich von 180 bis 199 liegen, ist gemeinsam, dass sie EIN Argument benoetigen, auch wenn es (wie im Falle von "POS" und "FRE") nicht beachtet wird, dessen Typ (numerisch oder String) festgelegt ist. Alle diese Funktionen liefern ausserdem EIN vom Typ her festgelegtes Ergebnis.

In der gleichen Weise funktioniert die "USR"-Funktion auch. Man gibt ein Argument mit und erhaelt ein Ergebnis. Der jeweilige Typ (sowohl Argument, als auch Ergebnis) wird durch das zugehoerige Maschinenprogramm festgelegt und kann nicht nur, wie dies besonders in Literatur zum PET und zu den CBMs immer wieder behauptet wird, im Integerbereich von -32767 bis 32767 liegen.

Die vom Betriebssystem festgelegte Startadresse der USR-Funktion ist 784 (siehe Vektortabelle der BASIC-Funktionen). Dort befindet sich aber der sogenannte USR-Vektor, also ein "JMP", dessen Argument vom Benutzer festzulegen ist. Im Normalfall zeigt das Argument dieses "JMP"-Befehls (Code 76, dezimal) auf die Fehlermeldung "?ILLEGAL QUANTITY ERROR", wodurch dieser Fehler auch bei Nichtsetzen des USR-Vektors und Benutzung des Befehls auftritt. Soll also das Maschinenprogramm zur Definition der USR-Funktion beispielsweise an der Adresse 828 (Cassettenpuffer) beginnen, so muss dies dem Computer durch ...

```
POKE 785, 60 : POKE 786, 3
```

... mitgeteilt werden. Die Reihenfolge der Abspeicherung ist natuerlich low/high. Nun zur Struktur von Funktionen im allgemeinen:

Das Betriebssystem wertet beim Erkennen einer Funktion (die Funktionen LEFT$, RIGHT$ und MID$ bilden eine Ausnahme, da sie mehrere Argumente benoetigen) das im Klammern stehende Argument, unabhaengig vom Typ des Arguments, aus. Sollte es sich bei dem Argument um einen numerischen Ausdruck handeln, so wird diese Zahl im Fliesskommaaccu #1 (Floatingpoint ACcumulator, FAC) abgelegt. Dabei handelt es sich um das Hauptrechenregister des Microsoft-BASICs. Dieses Argument kann dann durch Betriebssystemroutinen entsprechend weiterverarbeitet werden. Eine Unterscheidung zwischen Realzahlen und Integerzahlen wird hier nicht gemacht.

Handelt es sich jedoch um ein Argument des Typs String, so wird der Descriptor (bestehend aus drei Bytes, die die Laenge und Startadresse festlegen) dieses Strings in die Descriptortabelle gebracht. Durch die Routine "FRESTR" (Einsprungadresse 46755) wird zuerst geprueft, ob es sich bei dem Argument auch um einen String handelt. Daraufhin wird der durch den String belegte Speicherbereich wieder freigegeben und der Pointer (34/35) auf die Startadresse des Strings gesetzt. Der Accu enthaelt nach Rueckkehr die Stringlaenge.

Mit diesen Angaben kann nun die Auswertung beginnen: Als Beispiel soll hier eine mathematische Funktion dienen: die im Commodore-BASIC nicht implementierte Umkehrfunktion zur SINUS-Funktion. Die Umrechnungsformel lautet ...

ARC SIN (X) = ATN (X / SQR (1 - X * X))

Sollen von Maschinensprache aus Funktionen ausgewertet werden, so muss das Argument sich in FAC befinden. Nach Aufruf durch "JSR" befindet sich ebenfalls in FAC das Ergebnis. Fuer Operationen wie "-" oder "*" muss anders vorgegangen werden. Hier muss der erste Wert in das Rechenregister ARG (floatingpoint ARGument) uebertragen werden, der zweite Wert zur Ausfuehrung der Operation befindet sich (wie das Ergebnis der Operation auch) in FAC. Es muss also die Funktion zerlegt werden, aehnlich wie sie auf einem UPN-Taschenrechner eingegeben wuerde.

Nun die theoretische Vorgehensweise zur Berechnung dieser Funktion (das Argument "X" befindet sich in FAC):

"X" muss quadriert werden, also wird FAC in ARG kopiert und dann FAC mit ARG multipliziert, Ergebnis bereits in FAC.

Dieses Ergebnis muss nun von der Zahl "1" subtrahiert werden. Dazu wird die Konstante "1" in ARG uebertragen, davon wird FAC, der das Ergebnis der vorherigen Operation enthaelt, subtrahiert.

Daraus wird die Wurzel gezogen, was einfach durch Aufruf der "SQR"-Routine erfolgen kann. Nun muss das in FAC abgelegte Zwischenergebnis durch "X", also das urspruengliche Argument, geteilt werden. Da dieses Argument aber mittlerweile nicht mehr verfuegbar ist, muss es also zu Beginn des Maschinenprogramms in einen Bereich uebertragen werden, der durch keine bis hierhin verwendete Berechnung ueberschrieben wird. Dieses gerettete Argument wird sodann in ARG uebertragen und die Divisionsroutine aufgerufen.

Zum Abschluss erfolgt ein Sprung zur "ATN"-Routine, die dann noch den Arcustangens in FAC bringt. Da das Endergebnis einer Funktion sich in FAC befinden muss, ist keine weitere Behandlung des Ergebnis der "ATN"-Routine mehr noetig.

Hier nun das Maschinenprogramm zur Ausfuehrung der Berechnung:

```
 828 LDX      #0      (XR/YR) := 256
 830 LDY      #1
 832 JSR   48084      ) FAC nach 256 bis 260 retten
 835 JSR   48140      ) FAC nach ARG uebertragen
 838 LDA      97      Exponentbyte in Accu fuer Nullpruefung
 840 JSR   47659      ) FAC := ARG * FAC
 843 LDA    #188      (Accu/YR) := 47548
 845 LDY    #185      Startadresse der Konstanten "1"
 848 JSR   47184      ) FAC := Konstante / FAC
 851 JSR   49009      ) FAC := SQR (FAC)
 854 LDA      #0      (Accu/YR) := 256
 856 LDY      #1      Startadresse des geretteten Arguments
 858 JSR   47887      ) FAC := Konstante / FAC
 861 JMP   58126      ) FAC := ATN (FAC)
```

Das zugehoerige BASIC-Programm zum Einlesen der Maschinenroutine:

```
100 FOR I = 828 TO 862 : READ A : POKE I, A : NEXT
110 POKE 785, 60 : POKE 786, 3
120 DATA 162, , 160, 1, 32, 212, 187, 32, 12, 188, 165, 97
130 DATA 32, 43, 186, 169, 188, 160, 185, 32, 80, 184, 32
140 DATA 113, 191, 169, , 160, 1, 32, 15, 187, 76, 14, 227
```

Vielleicht ist es Ihnen aufgefallen, dass nicht der Typ des Arguments geprueft wurde. Dieser kann durch den Inhalt der Adresse 13 (siehe Memory Map) oder durch Aufruf der entsprechenden Routine zur Pruefung des Ausdrucktyps festgestellt werden. Dies ist jedoch nicht noetig, da die Adresse 13 durch Berechnungen nicht geaendert wird. Daher ist nach Beendigung der Berechnung diese Flag im Falle eines Stringarguments noch immer auf "String" gesetzt. Da jedoch bei dieser Funktion ein numerisches Ergebnis erwartet wird, erfolgt die Ausgabe von "?TYPE MISMATCH ERROR", falls ein Stringargument mitgegeben wurde.

Hat eine Funktion ein Ergebnis des Typs String, so muss anders vorgegangen werden, da eine direkte Rueckkehr zur aufrufenden Auswertungsroutine mittels "RTS" nur bei numerischem Resultat erfolgen darf. Zuerst muss fuer den Ergebnisstring der Funktion Platz im Arbeitsspeicher geschaffen werden. Zu diesem Zeitpunkt muss die Laenge des Ergebnisstrings bekannt sein, da bei Aufruf der Routine 46205 der Accu die Laenge enthalten muss. Entsprechend dieser Angabe wird dann Platz im Arbeitsspeicher geschaffen. Registerpaar (98/99) zeigt dann auf die Startadresse dieses Bereichs. Ist alles abgespeichert worden, so muss nun die Ruecksprungadresse in die Auswertungsschleife mittels "PLA", "PLA" vom Stack geholt werden. Durch "JMP 46282" wird dann der Descriptor des im oberen Stringbereich abgelegten Strings in die Descriptortabelle gebracht. Ein verstaendliches Beispiel fuer dieses Prinzip ist die Funktion "CHR$" ab der Adresse 46828.

STEUERCODES FUER DEN COMMODORE 64

000: -
001: -
002: -
003: RUNSTOP
004: -
005: WHITE: Cursorfarbe weiss
006: -
007: -
008: Shift + Commodore blockieren
009: Shift + Commodore freigeben
010: -
011: -
012: -
013: RETURN
014: Gross- und Kleinschrift
015: -
016: -
017: CURSOR DOWN: Cursor nach unten
018: REVERS ON: Negativdruck
019: HOME: Cursor nach oben links
020: DELETE: Zeichen loeschen
021: -
022: -
023: -
024: -
025: -
026: -
027: -
028: RED: Cursorfarbe rot
029: CURSOR RIGHT: Cursor nach rechts
030: GREEN: Cursorfarbe gruen
031: BLUE: Cursorfarbe blau

128: -
129: ORANGE: Cursorfarbe orange
130: -
131: Shift RUNSTOP
132: -
133: F1 Codes fuer die
134: F3 Funktionstasten
135: F5
136: F7
137: F2
138: F4
139: F6
140: F8
141: Shift RETURN
142: Grossschrift und Graphik
143: -
144: BLACK: Cursorfarbe schwarz
145: CURSOR UP: Cursor nach oben
146: REVERS OFF: Positivdruck
147: CLEAR SCREEN: Bildschirm loeschen
148: INSERT: Zeichen einfuegen
149: BROWN: Cursorfarbe braun
150: LT. RED: Cursorfarbe rosa
151: GRAY 1: Cursorfarbe dunkelgrau
152: GRAY 2: Cursorfarbe mittelgrau
153: LT. GREEN: Cursorfarbe hellgruen
154: LT. BLUE: Cursorfarbe hellblau
155: GRAY 3: Cursorfarbe hellgrau
156: PURPLE: Cursorfarbe purpur
157: CURSOR LEFT: Cursor nach links
158: YELLOW: Cursorfarbe gelb
159: CYAN: Cursorfarbe cyan

Nichtaufgefuehrte Codes haben in der Standardversion des Betriebssystems keinerlei Wirkung.

Saemtliche angegeben Steuercodes sind ueber die CHR$-Funktion sowie, von 142 abgesehen, auch ueber die Tastatur erreichbar.

Die Codes 013, 131, 141 und im Normalfall auch 020 und 148 werden immer ausgefuehrt und bewirken keine Ausgabe von Steuersymbolen. Sie sollten bei Verwendung in Programmen daher ueber der CHR$-Funktion programmiert werden.

Die Codes 003, 131 sowie 133 bis 140 (Funktionstasten) haben bei der Ausgabe keinerlei Funktion und finden daher im Normalfall wohl nur zur Abfrage der Tasten Verwendung. Lediglich der Code 131 hat ueber die Programmierung des Tastaturpuffers die Funktion, die er auch im Normalfall hat, das Einladen und Starten von Cassettenprogrammen.

Die Codes von 000 bis 031 sind unter anderem auch durch Druecken der Control-Taste zusammen mit einer der Tasten von "@", "A" bis "Z", ":", 'Pound', ";", "↑" und "=" erreichbar, es gelten jedoch auch hier die oben gemachten Einschraenkungen.

STEUERCODES IN LISTINGS

Waehrend sich beim PET 2001 die Anzahl der Steuercodes noch im Rahmen hielt, so gibt es beim Commodore 64 schon insgesamt 42 Codes, die irgendeine Bedeutung haben. Besonders beim Eintippen von Listings ist es notwendig, das Steuerzeichen mit der Taste (die diesen Code erzeugt) in Verbindung zu bringen. Deshalb eine Aufstellung aller verwendeten Steuercodes: Es wird jeweils das positive (zum besseren Erkennen) und das negative Zeichen (das in den Listings vorkommt) angegeben. Auch werden beide Darstellungsmodi (Graphik und Kleinschrift) beruecksichtigt:

```
C   c   - RUNSTOP
E   e   - Control E, Control 2
H   h   - Control H
I   i   - Control I
N   n   - Control N
Q   q   - Control Q, CURSOR DOWN
R   r   - Control R, Control 9
S   s   - Control S, HOME
T   t   - DELETE, CHR$(20)
£       - Control £, Control 3
]       - Control :, CURSOR RIGHT
↑       - Control ↑, Control 6
←       - Control =, Control 7
♠   A   - Commodore 1
─   C   - Shift-RUNSTOP
    E   - F1
    F   - F3
    G   - F5
    H   - F7
    I   - F2
    J   - F4
    K   - F6
L   L   - F8
    N   - CHR$(142)
    P   - Control 1
●   Q   - CURSOR UP
    R   - Control 0
♥   S   - CLEAR SCREEN
    T   - INSERT, CHR$(148)
    U   - Commodore 2
×   V   - Commodore 3
○   W   - Commodore 4
♣   X   - Commodore 5
    Y   - Commodore 6
♦   Z   - Commodore 7
┼       - Commodore 8
        - Control 5
        - CURSOR LEFT
π       - Control 8
        - Control 4
```

EINFUEHRUNG IN DIE ASSEMBLER-PROGRAMMIERUNG

Das "Herz" eines jeden Mikrocomputers ist der Mikroprozessor. Im VIC 20 ist ein Mikroprozessor des Typs "6502" eingebaut, im Commodore 64 ein "6510". Beide sind bezueglich der Programmierung in Assembler voellig identisch.

Jeder Mikroprozessor kann eine bestimmte Anzahl von genau definierten Assembler-Anweisungen verstehen und ausfuehren, etwa so, wie ein BASIC-Interpreter bestimmte BASIC-Befehle verarbeiten kann. Derjenige, der diese Assembler-Anweisungen beherrscht, kann also direkt den Mikroprozessor programmieren.

Programme, die in Assembler geschrieben sind, laufen wesentlich schneller ab als in BASIC und koennen die Moeglichkeiten des Computers viel besser ausnutzen. Fuer jeden fortgeschrittenen Programmierer ist daher die Beherrschung der Assembler-Sprache unerlaeßlich.

Um erfolgreich in Assembler programmieren zu koennen, ist es wichtig, den Aufbau des Computers zu kennen.
Jeder Computer besteht aus einer Vielzahl von kleinen Schaltern, die entweder offen sind (es fließt kein Strom) oder geschlossen (es fließt Strom). Natuerlich handelt es sich dabei nicht um mechanische Schalter, sondern vielmehr um Schaltelemente, die in IC-Bausteinen ("Chips") untergebracht sind. Einen solchen einzelnen Schalter nennt man "Bit" und statt "offen" und "geschlossen" redet der Fachmann von zwei "Zustaenden", die er "0" und "1" nennt. Ein solches System heißt "binaer", "zweier Zustaende faehig".

Der 6502/6510 Mikroprozessor ist derart aufgebaut, daß er jeweils 8 Bits gleichzeitig verarbeitet. Eine solche Einheit von 8 Bits nennen wir "Byte". Ein Beispiel fuer ein Byte ist "00110101".

```
    1 Byte = (z.B.)      0  0  1  1  0  1  0  1

    Bit Nummer           7  6  5  4  3  2  1  0
```

Wie Sie sehen, werden die einzelnen Bits eines Bytes von rechts nach links von 0 bis 7 durchnumeriert.

Nun ist der gesamte Computerspeicher aufgebaut aus einzelnen "Speicherzellen", wobei jede Zelle genau ein Byte groß ist. Sie koennen sich das ungefaehr so vorstellen:

```
  Speicherzelle 0:        1 0 1 0 1 0 1 0  = 8 Bit = 1 Byte
  Speicherzelle 1:        0 1 1 0 1 1 0 1
  Speicherzelle 2:        0 0 0 1 1 0 0 1
           .
           .
  Speicherzelle 65535:    0 1 0 0 0 0 1 1
```

Die Bytes unterscheiden sich dadurch voneinander, daß sie zum Teil unterschiedliche Bitkombinationen (z.B. "01000011") enthalten. Man redet auch davon, daß ein Bit "gesetzt" (= 1) oder "geloescht (= 0) ist. Die in der Skizze genannten Bitkombinationen sind willkuerlich gewaehlt.

Da 8 Bits nur 256 verschiedene Kombinationen darstellen koennen, kann der Mikroprozessor 6502/6510 grundsaetzlich nur Zahlen von 0 bis 255 bearbeiten (s. auch Kapitel "Binaerarithmetik").

```
Kombination   0:    0 0 0 0 0 0 0
Kombination   1:    0 0 0 0 0 0 1
Kombination   2:    0 0 0 0 0 1 0
Kombination   3:    0 0 0 0 0 1 1
Kombination   4:    0 0 0 0 1 0 0
        .
        .
Kombination 255:    1 1 1 1 1 1 1
```

Sie sehen, daß jeder Bitkombination, also jedem Byte, ein bestimmter dezimaler Wert von 0 bis 255 zugeordnet ist. Wenn Sie z.B. "POKE3,255" eingeben, so bedeutet das nichts anderes, als daß in die Speicherzelle 3 die Bitkombination "11111111" eingeschrieben wird.

Der Speicher besteht aus einzelnen Speicherzellen von der Groeße eines Bytes, die prinzipiell voellig gleich sind. Wir unterscheiden jedoch unterschiedliche Funktionsbereiche wie Programm- und Variablenspeicher, Betriebssystem- und BASIC-Interpreter-Bereich, Ein-/Ausgabespeicher, Betriebssystemzwischenspeicher u.a. Das vorliegende Buch erlaeutert die Arbeitsweise jedes einzelnen Funktionsbereiches im Detail. Dabei gibt es sog. "RAM-Speicher" und "ROM-Speicher". "RAM" bedeutet, daß Daten eingeschrieben und auch wieder gelesen werden koennen (z.B. Programm- und Variablenspeicher), "ROM" hingegen meint, daß der betreffende Speicherbereich ausschließlich gelesen, nicht aber durch anderen Daten ueberschrieben (veraendert) werden kann (z.B. BASIC-Interpreter).

Machen Sie sich klar, daß der gesamte Computerspeicher aus einzelnen Speicherzellen besteht, die wiederum Zahlen beinhalten. Diese Zahlen haben allerdings ganz verschiedene Bedeutungen, es kann sich um ein BASIC-Programm handeln, oder um ein Assembler-Programm, oder auch um Daten.

Sehen wir uns nun - ausgeruestet mit einem Minimum an theoretischen Wissen - das erste Assembler-Programm an. Um das Programm in den Computer eingeben zu koennen, benoetigen wir noch ein sog. "Assembler-System", das unsere Assembler-Anweisungen in reinen Zahlencode umsetzt. Sie finden ein solches - allerdings sehr einfaches Assembler-System - im Anschluß an dieses Kapitel. Es ist sicher sinnvoll, wenn Sie sich zunaechst dort mit der Bedienung vertraut machen. Sie koennen dann gleich die erklaerten Beispiele nachvollziehen.

Commodore 64	VC-20 (bis 8K)	VC-20 (groeßer 8K)
828 LDA #65	828 LDA #65	828 LDA #65
830 STA 1024	830 STA 7680	830 STA 4096
833 LDA #3	833 LDA #3	833 LDA #3
835 STA 55296	835 STA 38400	835 STA 37888
838 RTS	838 RTS	838 RTS

Um das Programm auszuprobieren, geben Sie von BASIC aus SYS 828 (RETURN) ein. Es sollte sodann ein Zeichen in der linken oberen Bildschirmecke zu sehen sein.

Jeder Assembler-Schritt besteht aus:

828	LDA	#65
Adresse	Assemblerbefehlswort ("auch Mnemonic")	Argument

Die "Adresse" gibt an, ab welcher Speicherzelle die Assembler-Anweisung beginnt. Es existieren insgesamt 65536 verschiedene Speicherzellen, die die Adressen von 0 bis 65535 tragen. Jede Assembler-Anweisung belegt je nach Befehl 1 bis 3 Bytes. Der Adresse folgt das eigentliche Assembler-Befehlswort, im Beispiel "LDA". Sie finden in diesem Buch ein Verzeichnis aller gueltigen Befehlsworte. Dem Befehlswort kann - je nach Befehl - noch ein Argument folgen, in unserem Beispiel "Doppelkreuz 65". Es gibt auch Assembler-Anweisungen ohne Argument. Schließlich existieren ja auch in BASIC Befehle mit und ohne Argument (z.B. POKE mit zwei Argumenten, STOP ohne Argument).

Das Befehlswort "LDA" steht fuer "load accumulator", "Lade Akkumulator", oder in diesem Fall "Lade Akkumulator mit dem Wert 65". "STA" bedeutet "store accumulator", "speichere Akkumulator an der angegebenen Adresse ab".

Um das verstehen zu koennen, muessen Sie wissen, daß es in Assembler im Unterschied zu BASIC keine Variablen gibt. Als Ersatz dafuer existieren in Asembler drei "Register", naemlich "Akkumulator", "X-Register" und "Y-Register". Ein solches Register kann Werte von 0 bis 255 beinhalten. Jetzt koennen Sie das Programm verstehen (beispielhaft fuer den Commodore 64):

```
828 LDA #65        ;Akkumulator mit dem Wert 65 laden
830 STA 1024       ;diese 65 in Speicherzelle 1024 speichern
833 LDA #3         ;Akkumulator mit dem Wert 3 laden und
835 STA 55296      ;die 3 in Speicherzelle 55296 speichern
838 RTS            ;beendet jedes Assembler-Programm
```

Beachten Sie die Bedeutung des Doppelkreuz-Zeichens: Der Akkumulator wird direkt mit dem Wert 65 geladen (und nicht mit dem Wert, der in der Speicherzelle 65 steht, was auch moeglich waere). Hingegen wird diese 65 dann in Speicherzelle 1024 abgespeichert. Die Anweisung RTS ("return from subroutine", "Rueckkehr aus einer Unterroutine") muß unbedingt am Ende jedes Assemblerprogramms stehen. Andernfalls geraet der Computer im allgemeinen in einen voellig unkontrollierten Zustand ("Absturz"), da er versucht, weitere Anweisungen auszufuehren, obwohl kein gueltiger Assemblercode mehr vorhanden ist.

Warum unser Programm ein Zeichen auf dem Bildschirm ausgibt? Nun, Speicherzelle 1024 (Commodore 64) entspricht der linken oberen Bildschirmposition (Zeichencode), Speicherzelle 55296 bestimmt die Farbe eben dieser Position. Experimentieren Sie selbstaendig in dieser Richtung, indem Sie statt 65 andere Werte zwischen 0 und 255 nehmen, statt der 3 andere Werte von 0 bis 7 (VIC 20) bzw. von 0 bis 15 (Commodore 64).

```
Assembler                  BASIC

 828 LDA #65                828 A = 65
 830 STA 1024               830 POKE 1024, A
 833 LDA #3                 833 A = 3
 835 STA 55296              835 POKE 55296, A
 838 RTS                    838 END
```

Natuerlich existieren in Assembler noch eine Vielzahl anderer Befehlsworte. Ein Teil davon ist den bereits gelernten so aehnlich, daß Sie sie sofort verstehen werden.

Fuer das X- und Y-Register lauten die Befehlsworte zum Laden und Abspeichern wie folgt:

Assembler	Funktion	BASIC-Entsprechung
LDX #wert	laedt X-Register	X = wert
STX adresse	speichert X-Register	POKE adresse, X
LDY #wert	laedt Y-Register	Y = wert
STY adresse	speichert Y-Register	POKE adresse, Y

X- und Y-Register sind - genau wie der Akkumulator - interne Register des Mikroprozessors von der Groeße eines Bytes. Sie koennen also Zahlen von 0 bis 255 enthalten.

Um einen Text auf dem Bildschirm auszugeben, wird dieser einfach Zeichen fuer Zeichen mit LD? (LDA, LDX oder LDY) in ein Register geladen und mit ST? an die entsprechende Bildschirmposition gebracht. Außerdem muß die korrespondierende Speicherzelle des Farbspeichers mit einem Farbwert versehen werden (s. auch Kapitel "Bildschirmspeicher" und "Farbspeicher").

Weitere wichtige Assembler-Befehle in Bezug auf die Prozessorregister sind Anweisungen zum Erhoehen und Erniedrigen der Register:

Assembler	Funktion	BASIC-Entsprechung
INX	erhoeht X-Reg. um eins	X = X + 1
DEX	erniedrigt X-Reg. um eins	X = X - 1
INY	erhoeht Y-Reg. um eins	Y = Y + 1
DEY	erniedrigt Y-Reg. um eins	Y = Y - 1

Diese Anweisungen benoetigen kein Argument. Ein entsprechender Befehl zum Inkrementieren oder Dekrementieren des Akkumulators existiert nicht. Hingegen gibt es die Moeglichkeit, einzelne Speicherzellen direkt um eins zu erhoehen oder zu erniedrigen.

Assembler	BASIC-Entsprechung
INC adresse	POKE adresse, PEEK(adresse) + 1
DEC adresse	POKE adresse, PEEK(adresse) - 1

Ein Beispielprogramm soll Ihnen die Moeglichkeiten der Inkrement/Dekrement-Anweisungen verdeutlichen (Commodore 64):

```
Assembler          Funktion                BASIC-Entsprechung

830 LDX #0         ;Schleifenzaehler        830 X = 0
832 LDA #42        ;Zeichencode "*"         832 A = 42
834 STA 1024,X     ;Bildschirmspeicher      834 POKE 1024+X,A
837 LDA #3         ;Farbzahl                837 A = 3
839 STA 55296,X    ;Farbspeicher            839 POKE 55296+X,A
842 INX            ;X-Reg. von 0 bis        842 X = X + 1
843 CPX #50        ;50 zaehlen und
845 BNE 832        ;Sprung zu 832           845 IFX  50THEN832
847 RTS            ;Programmende            847 END
```

Anpassung fuer VIC 20: bis 8K: 7680 statt 1024, 38400 statt 55296; ueber 8K: 4096 statt 1024, 37888 statt 55296. Geben Sie das Programm ein und starten Sie es mit SYS 830. Es sollten 50 farbige Sternchen zu sehen sein.

Experimentieren Sie selbstaendig: Ersetzen Sie einmal die 42 durch andere Werte von 0 bis 255, die 3 durch andere Werte von 0 bis 7 (VIC 20) bzw. von 0 bis 15 (Commodore 64), die 50 durch andere Werte von 0 bis 255.

Zur Erklaerung: Mit Hilfe des X-Registers wird eine Schleife programmiert, die von 0 bis 50 zaehlt. INX bewirkt jeweils die Erhoehung um eins. "CPX" steht dabei fuer "compare x-register", "vergleiche X-Register", in diesem Fall mit 50. Darauf folgt die BNE-Anweisung. "BNE" bedeutet "branch on not equal", "verzweige bei Ungleichheit". Mit anderen Worten: Solange das X-Register noch nicht gleich 50 ist, wird zu Adresse 832 zurueckgesprungen. CPX und BNE hintereinander entsprechen in BASIC einer IF-THEN-Anweisung (bedingte Sprunganweisung mit logischem Vergleich).

Insgesamt existieren in Assembler drei Vergleichsbefehle:

Assembler	Funktion
CMP #wert	vergleicht Akkumulator mit dem Wert
CPX #wert	vergleicht X-Register mit dem Wert
CPY #wert	vergleicht Y-Register mit dem Wert

Entsprechungen in BASIC gibt es nicht. Auf jede Vergleichsanweisung folgt ein sog. "Branch-Befehl", in unserem Beispiel ist das "BNE". Das ist ungefaehr so, wie in BASIC auf IF in jedem Fall THEN folgen muß.

Damit existieren in Assembler die folgenden Moeglichkeiten bedingter Sprunganweisungen:

Sprung bei Gleichheit:	CMP #wert BEQ adresse
Sprung bei Ungleichheit:	CMP #wert BNE adresse
Sprung bei 'kleiner als':	CMP #wert BCC adresse
Sprung bei 'groesser/gleich':	CMP #wert BCS adresse

Wir moechten allerdings an dieser Stelle ausdruecklich darauf hinweisen, dass die Erklaerungen im Rahmen dieser "Einfuehrung in die Assembler-Programmierung" stark vereinfacht und keineswegs vollstaendig sind. Ein umfassender systematischer Assembler-Kurs wuerde den Rahmen dieses Buches sprengen. Allen wirklich Interessierten empfehlen wir unseren "6502-Assembler-Kurs fuer Beginner", ISBN 3-88986-000-1. Schließlich ist ein Systemhandbuch weder ein BASIC- noch ein Assembler-Lehrbuch.

Soll ein Sprung unabhaengig von irgendwelchen Bedingungen erfolgen, so wird dies in Assembler mit dem Befehlen JMP (normaler Sprung) oder JSR (Unterprogrammsprung) erreicht. "JMP" steht fuer "jump", "Sprung", "JSR" bedeutet "jump to subroutine", "Sprung zu Unterprogramm".

Assembler	Funktion	BASIC-Entsprechung
JMP adresse	Sprung	GOTO zeilennummer
JSR adresse	Unterprogrammsprung	GOSUB zeilennummer

Die Rueckkehr aus einem Unterprogramm erfolgt mittels RTS, "return from subroutine".

Assembler	Funktion	BASIC-Entsprechung
RTS	Ruecksprung aus Unterprogramm	RETURN

Es wird Ihnen auffallen, daß dies derselbe Befehl ist, mit dem jedes Assembler-Programm enden muß. Der Grund dafuer ist recht einfach: Von BASIC aus wird jedes Assembler-Programm als Unterprogramm aufgerufen (SYS, USR), folglich muß es auch wie ein Unterprogramm abgeschlossen werden.

Wie Sie bereits wissen, gibt es drei Arbeitsregister in der CPU 6502/6510: Akkumulator sowie X- und Y-Register. Der Transfer von Daten zwischen diesen Registern erfolgt mit den folgenden Anweisungen:

Assembler	Funktion: transferiert...	BASIC-Entsprechung
TAX	Akku nach X-Register	X = A
TAY	Akku nach Y-Register	Y = A
TXA	X-Register nach Akku	A = X
TYA	Y-Register nach Akku	A = Y

Beispiel: Im Rahmen eines groesseren Programmes sollen Akkumulator sowie X- und Y-Register mit dem Wert der Speicherzelle 900 geladen werden (Adressen/Zeilennummern wurden weggelassen):

Assembler	BASIC
LDA 900	A = PEEK(900)
TAX	X = A
TAY	Y = A

Beachten Sie, daß diesmal LDA ohne Doppelkreuz verwendet wird, da hier nicht der Wert 900 gemeint ist, sondern die Speicherzelle mit der Adresse 900.

Zwei weitere wichtige Befehle zum Addieren und Subtrahieren in Assembler sollten Sie noch kennenlernen. Die Befehlsworte lauten "ADC", "add with carry", "Addiere mit Uebertrag" und "SBC", "subtract with carry", "Subtrahiere mit Uebertrag". Wichtig in diesem Zusammenhang ist das sog. "Carry-Bit" oder "Carry-Flag" zur Bestimmung des Uebertrages (Zum Begriff "Flag" s. auch Kapitel "Binaerarithmetik"). Der Vorgang ist im Grunde recht einfach: Der Mikroprozessor 6502/6510 kann - wie Sie wissen - nur Zahlen von 0 bis 255 verarbeiten. Daher muß vor jeder Addition das Carry-Bit geloescht (= 0) werden. Dann werden die beiden Zahlen addiert und ist das Ergebnis groesser als 255, so wird das Carry-Bit gesetzt (= 1), andernfalls bleibt es geloescht.

```
828 LDA #24     ;laedt Akku mit dem Wert 24
830 CLC         ;loescht Carry-Bit (CLC, "clear carry")
831 ADC #68     ;addiert den Wert 68 zu dem Wert 24
833 STA 850     ;speichert Ergebnis (92) nach Adresse 850
836 RTS         ;Programmende
```

Das Ergebnis der Addition kann von BASIC aus mit ?PEEK(850) abgefragt werden; gestartet wird das Programm mit SYS 828.

Die Subtraktion wird in Assembler aehnlich programmiert, nur darf dabei das Carry-Bit vorher nicht geloescht, sondern muss im Gegenteil gesetzt werden:

```
828 LDA #128       ;laedt Akku mit dem Wert 128
830 SEC            ;setzt Carry-Bit (SEC, "set carry")
831 SBC #6         ;subtrahiert den Wert 6 von 128
833 STA 850        ;speichert Ergebnis(122) nach Adresse 850
836 RTS            ;Programmende
```

Ist das Ergebnis der Subtraktion im "normalen" Bereich von 0 bis 255, so bleibt das Carry-Bit unveraendert gesetzt, andernfalls wird es geloescht.

Mit BCC und BCS laeßt sich der Zustand des Carry-Bits feststellen. "BCC" bedeutet "branch on carry clear", "Sprung bei geloeschtem Carry-Bit", "BCS" hingegen "branch on carry set", "Sprung bei gesetztem Carry-Bit". Sie kennen die Anweisungen BCC und BCS bereits aus dem Zusammenhang "Bedingte Sprunganweisungen".

```
828 LDA #128       ;laedt Akku mit dem Wert 128
830 CLC            ;setzt Carry-Bit (SEC, "set carry")
831 ADC #200       ;addiert den Wert 200 zu dem Wert 128
833 BCS 838        ;Sprung nach 838, wenn Carry-Bit gesetzt
835 STA 850        ;speichert Ergebnis nach Adresse 850
838 RTS            ;Programmende
```

In diesem Beispiel wird das Ergebnis nur dann in Speicherzelle 850 abgespeichert, wenn es im zulaessigen Bereich von 0 bis 255 liegt. Andernfalls wird durch die Addition mit ADC das Carry-Bit gesetzt (Ueberlauf) und die danach folgende BCS-Anweisung verzweigt den Programmablauf zu Adresse 838, die STA-Anweisung bei 835 wird also uebersprungen. Der Wert in Speicherzelle 850 bleibt in diesem Fall unveraendert. Mit den Beispielwerten 128 und 200 wird dies natuerlich immer der Fall sein - schliesslich ist 328 ja groesser als 255. Probieren Sie einmal andere Werte aus.

Außer dem Carry-Bit oder Carry-Flag - auch "C-Flag" - gibt es noch sechs andere sog. "Statusbits" oder "Statusflags". Alle sieben Statusbits sind im Prozessor zusammengefasst im "Prozessor-Statusregister". Das achte Bit dieses Registers wird nicht genutzt. Im einzelnen heissen die Status-Bits:

N: negativ result (negatives Ergebnis)
V: overflow (Vorzeichen-Ueberlauf)
B: break command (Break-Befehl)
D: decimal mode (Dezimalbetrieb)
I: interrupt disable (Interrupt sperren)
Z: zero result (Null-Ergebnis)
C: Carry

Es wuerde den Rahmen dieses Einfuehrungskurses sprengen, wollten wir alle Flags erklaeren. Die zwei am haeufigsten anzutreffenden sind N- und Z-Flag.
Das Z-Flag wird immer dann vom Prozessor selbst gesetzt, wenn das Ergebnis des direkt vorausgegangenen Befehls Null ist. Andernfalls wird das Z-Flag geloescht. Ausschnitt aus einem groesseren Programm:

```
                   ;laedt Akku mit dem Inhalt von Speicher-
900 LDA 1024       ;zelle (Adresse) 1024
903 BEQ 920        ;Sprung zu 920, wenn Akkuinhalt gleich 0
```

BEQ fragt das Z-Flag ab und bewirkt einen Sprung zu der angegebenen Adresse (920), wenn es gesetzt ist. Das wiederum haengt davon ab, ob das Ergebnis der LDA-Operation Null ist oder nicht. BNE ist das Gegenstueck zu BEQ; bei BNE erfolgt der Sprung dann, wenn das Z-Flag geloescht ist. Sie haben BEQ und BNE bereits im Zusammenhang mit Vergleichsanweisungen kennengelernt.

Hier die Befehle zur Abfrage des Prozessor-Statusregisters:

```
BCC  - Sprung bei geloschtem Carry-Flag
BCS  - Sprung bei gesetztem Carry-Flag
BEQ  - Sprung bei gesetztem Z-Flag
BNE  - Sprung bei geloeschtem Z-Flag
BMI  - Sprung bei gesetztem N-Flag
BPL  - Sprung bei geloeschtem N-Flag
BVC  - Sprung bei geloeschtem V-Flag
BVS  - Sprung bei gesetztem V-Flag
```

Zum Abschluß wollen wir uns jetzt noch ansehen, wie eigentlich Assembler-Programme im Speicher abgelegt werden. Das ist naemlich fuer das Grundverstaendnis der Programmierung in Assembler recht wichtig.

Der Mikroprozessor ist prinzipiell nur in der Lage, Zahlen zu verarbeiten. Er muß sowohl Daten als auch Steueranweisungen als Zahlen erhalten. Deswegen wird jedes Assembler-Programm als eine Folge von Zahlen abgespeichert. Wenn Sie Assembler-Text wie "LDA, "STX" etc. eingeben, so sorgt das betreffende Assembler-System dafuer, dass dieser Text bei Druecken der RETURN-Taste sofort in eine entsprechende Zahlenfolge zerlegt und abgespeichert wird. Daher ist zur Assembler-Programmierung immer auch ein Assembler-System notwendig - egal, ob Sie nun das in diesem Buch abgedruckte einfache System oder ein so komplexes Assembler-System wie 'T.EX.AS.' - "Terminal Extended Assembler" - benutzen.

Assemblertext	Zahlencode
828 LDA #65	169 65
830 STA 1024	141 0 4
833 LDA #3	169 3
835 STA 55296	141 0 216
838 RTS	96

Das Programm steht wie folgt im Speicher:

Adresse	Inhalt (dezimal)	Bedeutung
828	169	LDA #
829	65	Argument zu LDA #
830	141	STA
831	0	1. Teilargument zu STA
832	4	2. Teilargument zu STA
833	169	LDA #
834	3	Argument zu LDA #
835	141	STA
836	0	1. Teilargument zu STA
837	216	2. Teilargument zu STA
838	96	RTS

Die Zerlegung einer Zahl in zwei Teilzahlen - genannt LSB (1. Teilzahl) und MSB (2. Teilzahl) - erfolgt so:

Gesamtzahl = LSB + 256 x MSB

Beispiel: 1048 geteilt durch 256 ergibt 4,09375. 4 mal 256 ist gleich 1024. 1048 minus 1024 ergibt 24. Also ist das MSB gleich 4 und das LSB gleich 24. Denn:

1024 = 24 + 256 x 4

Die Zerlegung einer Zahl in zwei Teilzahlen ist notwendig, da eine Speicherzelle nur Zahlen von 0 bis 255 aufnehmen kann. Groessere Zahlen werden in zwei aufeinanderfolgenden Speicherzellen eben in zwei Teilzahlen als LSB und MSB dargestellt. "LSB" bedeutet "least significant byte", "niederwertiges Byte", "MSB" steht fuer "most significant byte", "hoeherwertiges Byte" (s. auch Kapitel "Binaerarithmetik").

Die Zuordnung der Assembler-Befehlsworte zu Zahlen ist standardmaessig festgelegt (STA ist gleich 141 usw.). Neben dem Befehlswort spielt dabei auch die "Adressierungsart" eine Rolle. Sie haben bereits folgende Adressierungsarten kennengelernt:

```
unmittelbar         LDA #wert
absolut             LDA adresse
                    STA adresse
                    LDX adresse
                    STX adresse
                    LDY adresse
                    STY adresse
                    JMP adresse
                    usw.
implizit            INX
                    DEX
                    INY
                    DEY
                    RTS
                    usw.
indiziert           STA adresse,X
                    STA adresse,Y
                    usw.
```

Insgesamt gibt es in Assembler 13 verschiedene Adressierungsarten. Die Zuordnung aller Assembler-Befehlsworte in den moeglichen Adressierungsarten zu den entsprechenden Zahlencodes finden Sie in diesem Buch in der Tabelle "Adressierungsarten".

Damit sind wir am Ende dieses "Assembler-Schnellkurses" angelangt. Sie sind - falls Sie nicht noch andere Literatur zu Rate gezogen haben - jetzt weder in der Lage ein komplexeres Assembler-Programm zu lesen noch gar eines zu schreiben. Aber Sie verstehen genug von Assembler, um erstens zu wissen, worum es dabei geht und zweitens sich entscheiden zu koennen, ob Sie sich naeher damit befassen moechten. In diesem Fall moechten wir Ihnen als ausfuehrliches und leicht verstaendliches Lehrbuch nochmals den "6502-Assemblerkurs fuer Beginner", ISBN 3-88986-000-1, empfehlen. Das Buch nimmt speziell Bezug auch auf die Computer Commodore 64 und VIC 20. Zur fortgeschrittenen Programmierung in Assembler offerieren wir Ihnen unser "'T.EX.AS.' - Assembler Entwicklungs- und Lehrsystem". Wir wuerden uns jedenfalls freuen, Sie in dem einen oder anderen Werk wieder begruessen zu koennen. INTERFACE AGE VERLAG GMBH, Vohburgerstr. 1, D-8000 Muenchen 21, Tel. (089) 5 80 67 02.

ASSEMBLER UND DISASSEMBLER FUER COMMODORE 64 UND VIC-20

Um das Erstellen eigener Maschinenprogramme zu vereinfachen, ist im folgenden ein einfacher, zum groessten Teil in BASIC geschriebener Assembler, abgedruckt. Mit diesem ist es moeglich, Maschinenprogramme, Bytes und Adressen einzugeben, wobei der Assembler alle notwendigen Umwandlungen selbst vornimmt. Auch koennen beliebige Speicherbereiche disassembliert werden. Hierzu die genaue Anleitung zur Handhabung dieses Assemblers:

Beim Eintippen des Assemblers muss ganz besonders auf die korrekte Eingabe der DATA-Zeilen geachtet werden. Daher wird empfohlen, den gesamten Text nach der Eingabe noch einmal zu kontrollieren. Auch sollte man den Assembler vor dem ersten Start durch "RUN" erst einmal auf Cassette oder Disc abspeichern, um sich ein zweites Eintippen im Falle eines "Absturzes" zu ersparen.

Ist der Assembler durch "RUN" gestartet worden, so muss sich nach einigen Sekunden der Computer mit einem blinkenden Cursor melden. Nun ist der Assembler eingabebereit und erwartet Ihre Anweisungen.

Jede Eingabezeile MUSS mit einer Adresse im Bereich von 0 bis 65535 beginnen. Bei folgender Eingabe eines Maschinenbefehls ist dies die Adresse, ab der der Befehl (gegebenenfalls mit Argument) abgelegt werden soll. So zum Beispiel bei Eingabe des folgenden Beispielprogramms fuer den 64er:

```
828 LDX #0
830 LDA 646
833 STA 55296,X
836 TXA
837 STA 1024,X
840 INX
841 BNE 830
843 RTS
```

Wie Sie sicher bemerken, wird automatisch die naechste Adresse nach dem eingegebenen Befehl vorgegeben, aehnlich einem Autonumber fuer BASIC. Auch muss bei Verzweigungsbefehlen nicht der Offset angegeben werden, sondern die Absolutadresse.

Entsprechend erfolgt die Eingabe von Bytes und Adressen. Auf die Adresse folgt das Byte beziehungsweise die Adresse, die an der angegebenen Adresse abgelegt werden soll. So entspricht ...

```
650 128
```

... dem Befehl "POKE 650, 128". Fuer die Eingabe von Adressen existiert kein gleichwertiger Befehl im Commodore-BASIC, lediglich EXBASIC besitzt den Befehl "DOKE". So koennte von EXBASIC aus statt ...

```
785 828
```

... auch geschrieben werden "DOKE 785, 828" oder in diesem Fall "DEF USR = 828". Soll ein Wert im Bereich von 0 bis 255 als Pointer abgelegt werden, so muss vor den Wert der Klammeraffe "@" gesetzt werden, um dem Computer zu zeigen, dass es sich um einen 16-Bit-Wert handelt.

Fuer die Ausgabe von Speicherinhalten stehen drei Befehle zur Auswahl: "D" fuer Daten in Form von Bytes, "A" fuer Adressen und "L" fuer Assemblertext. Die Syntax fuer alle diese Befehle ist gleich:

Startadresse Befehl (Endadresse)

Um den Speicherbereich von 41866 bis 41900 zu disassemblieren, muss eingegeben werden ...

41866 L 41900

Leerzeichen koennen, ausser bei der Eingabe von Bytes und Adressen (zur Trennung der Zahlen) entfallen.

Das Verlassen des Assembler erfolgt durch das Kommando "X", wobei auch hier eine Zahl vor dem Buchstaben stehen muss (die allerdings keine Bedeutung hat).

Hinweise fuer VIC-20

Fuer den VIC-20 wird mindestens eine Erweiterung um 3 KB benoetigt, um mit dem Assembler arbeiten zu koennen. Da jedoch beim VIC-20 kein freier Bereich fuer das Maschinenprogramm zur Verfuegung steht, muss der Anfang des BASIC-Bereichs verlegt werden. Dies muss VOR der Eingabe des Listings sowie VOR dem Einladen geschehen.

VICs mit INSGESAMT 8 KB, also mit 3-KB-Erweiterung:

Verlagerung des BASIC-Anfangs:

POKE 44, 5 : POKE 1280, 0

Programmaenderungen (Position im Listing unterstrichen):

```
110 FORI=1025TO1240:...        120 SYS1117:...
470 DATA...11,4...             490 DATA...11,4...
```

Aufhebung der Verlagerung:

POKE 44, 4 : NEW

VICs mit MEHR ALS 8 KB, also mit 8 oder 16-KB-Erweiterung:

Verlagerung des BASIC-Anfangs:

POKE 44, 19 : POKE 4608, 0

Programmaenderungen (Position im Listing unterstrichen):

```
110 FORI=4609TO4824:...        120 SYS4701:...
470 DATA...11,18...            490 DATA...11,18...
```

Aufhebung der Verlagerung:

POKE 44, 18 : NEW

Die Aufhebung der Verlagerung ist notwendig, wenn andere Programme geladen werden sollen oder BASIC programmiert werden soll.

Ausserdem muss bei allen VICs vor den Befehl "SPC(20)" in Zeile 370 der Leerstring ("") gesetzt werden.

```
100 A$="":B$="":I=.:J=.:A=.:P=.:Q=.:B=.:F=.:DEFFND(I)=PEEK(I)+256*PEEK(I+1)
110 FORI=53032TO53247:READA:POKEI,A:NEXT:DIMB$(255):FORI=.TO191:READB$(I):NEX
120 SYS53124:A$=A$+"":IFA$=""GOTO120
130 A=FND(253):P=FND(251):I=.:IFLEN(A$)>2GOTO300
140 B=ASC(A$):IFB=91THENPOKEA,P:P=A:GOTO170
150 IFB=93THENPOKEA,PEEK(251):POKEA+1,PEEK(252):P=A:GOTO190
160 IFB-68GOTO180
170 GOSUB370:B=1:Q=PEEK(A):GOSUB390:PRINT:ONFGOTO170:GOTO290
180 IFB-65GOTO200
190 GOSUB370:B=2:Q=FND(A):GOSUB390:PRINT:ONFGOTO190:GOTO290
200 IFB-76THENON-(B<>88)GOTO120:END:GOTO120
210 GOSUB370:I=PEEK(A):IFI-4*INT(I/4)=.75THENI=2
220 A$=B$(I*.75+.75):PRINTLEFT$(A$,3)" ";:B=2:IFLEN(A$)=3THENB=1:GOTO280
230 FORJ=4TOLEN(A$):B$=MID$(A$,J,1):IFB$="["THENQ=PEEK(A+1):GOSUB360:GOTO270
240 IF(IAND31)=16THENPOKE144,PEEK(A+1):Q=ST+A+2:GOSUB360:GOTO270
250 IFB$="]"THENQ=FND(A+1):B=3:GOSUB360:GOTO270
260 PRINTB$;
270 NEXT
280 PRINT:GOSUB400:ONFGOTO210
290 GOSUB380:PRINT"]":FORI=.TO5:POKE631+I,29:NEXT:POKE198,6:GOTO120
300 ON-(B$(I)=A$)GOTO310:I=I+1:ON-(I<192)GOTO300:GOTO120
310 I=INT(I/3)+I:B=3:IFLEN(A$)=3THENB=1:GOTO340
320 IFMID$(A$,4,1)="["ORMID$(A$,5,1)="["THENB=2
330 IF(IAND31)=16THENB=2:P=P-A-2:ON-(P<-128ORP>127)GOTO120:POKE251,PAND255
340 POKEA,I:IFB>1THENPOKEA+1,PEEK(251):IFB=3THENPOKEA+2,PEEK(252)
350 P=A:GOTO210
360 PRINTMID$(STR$(Q),2);:RETURN
370 POKE211,.:OPEN3,3:PRINT#3,SPC(20):CLOSE3:POKE211,.
380 PRINTRIGHT$("    "+STR$(A),5)" ";:RETURN
390 PRINTMID$(STR$(Q),2);
400 A=A+B:A=A+65536*(A>65535):F=1-ABS(SGN(PEEK(653)ORA>P)):RETURN
410 DATA24,165,7,5,252,133,7,96,56,96,169,,133,251,133,252,133,7,202,232,189,
420 DATA2,201,32,240,248,232,201,64,240,240,202,201,48,144,227,201,58,176,224
430 DATA189,,2,56,233,48,144,208,201,10,176,203,72,165,252,133,8,165,251,10,3
440 DATA8,10,38,8,101,251,133,251,165,8,101,252,133,252,6,251,38,252,104,101
450 DATA251,133,251,144,2,230,252,232,208,205,169,,133,211,133,208,32,207,255
460 DATA201,32,240,249,162,,201,13,240,13,157,,2,134,7,32,207,255,166,7,232
470 DATA208,239,138,240,221,169,,157,,2,168,170,32,50,207,165,251,133,253,165
480 DATA252,133,254,176,201,202,232,189,,2,240,46,201,32,240,246,201,65,176,4
490 DATA201,48,176,6,153,,1,200,208,232,32,50,207,176,23,8,169,91,40,240,2
500 DATA169,93,153,,1,200,189,,2,240,6,153,,1,232,208,244,152,160,2,145,45,16
510 DATA,200,145,45,169,1,200,145,45,96
520 DATABRK,"ORA([,X)",,,ORA[,ASL[,PHP,ORA#[,ASL,,ORA],ASL],BPL],"ORA([),Y
530 DATA,,"ORA[,X","ASL[,X",CLC,"ORA],Y",,,"ORA],X","ASL],X",JSR],"AND([,X)
540 DATA,BIT[,AND[,ROL[,PLP,AND#[,ROL,BIT],AND],ROL],BMI],"AND([),Y",,,"AND[,
550 DATA"ROL[,X",SEC,"AND],Y",,,"AND],X","ROL],X",RTI,"EOR([,X)",,,EOR[,LSR[
560 DATAPHA,EOR#[,LSR,JMP],EOR],LSR],BVC],"EOR([),Y",,,"EOR[,X","LSR[,X",CLI
570 DATA"EOR],Y",,,"EOR],X","LSR],X",RTS,"ADC([,X)",,,ADC[,ROR[,PLA,ADC#[,ROR
580 DATAJMP(]),ADC],ROR],BVS],"ADC([),Y",,,"ADC[,X","ROR[,X",SEI,"ADC],Y",,
590 DATA"ADC],X","ROR],X",,"STA([,X)",,STY[,STA[,STX[,DEY,,TXA,STY],STA],STX]
600 DATABCC],"STA([),Y",,"STY[,X","STA[,X","STX[,Y",TYA,"STA],Y",TXS,,"STA],X
610 DATA,LDY#[,"LDA([,X)",LDX#[,LDY[,LDA[,LDX[,TAY,LDA#[,TAX,LDY],LDA],LDX]
620 DATABCS],"LDA([),Y",,"LDY[,X","LDA[,X","LDX[,Y",CLV,"LDA],Y",TXS,"LDY],X
630 DATA"LDA],X","LDX],Y",CPY#[,"CMP([,X)",,CPY[,CMP[,DEC[,INY,CMP#[,DEX,CPY]
640 DATACMP],DEC],BNE],"CMP([),Y",,,"CMP[,X","DEC[,X",CLD,"CMP],Y",,,"CMP],X
650 DATA"DEC],X",CPX#[,"SBC([,X)",,CPX[,SBC[,INC[,INX,SBC#[,NOP,CPX],SBC],INC
660 DATABEQ],"SBC([),Y",,,"SBC[,X","INC[,X",SED,"SBC],Y",,,"SBC],X","INC],X
```

BEFEHLSLISTE

ADC: Speicher mit Carry zu Accumulator addieren
AND: Speicher durch "AND" mit Accumulator verknuepfen
ASL: Speicher/Accumulator um ein Bit linksverschieben
BCC: Verzweigung bei geloeschtem Carry
BCS: Verzweigung bei gesetztem Carry
BEQ: Verzweigung bei gesetzter Zero-Flag
BIT: Bit-Test zwischen Speicher und Accumulator
BMI: Verzweigung bei gesetzter Negative-Flag
BNE: Verzweigung bei geloeschter Zero-Flag
BPL: Verzweigung bei geloeschter Negative-Flag
BRK: Ausloesung eines Software-Interrupts
BVC: Verzweigung bei geloeschter Overflow-Flag
BVS: Verzweigung bei gesetzter Overflow-Flag
CLC: Carry loeschen
CLD: Dezimalmodus ausschalten
CLI: Interrupt freigeben
CLV: Overflow-Flag loeschen
CMP: Speicher mit Accumulator vergleichen
CPX: Speicher mit X-Register vergleichen
CPY: Speicher mit Y-Register vergleichen
DEC: Verminderung des Speicherinhalts um eins
DEX: Verminderung des X-Registers um eins
DEY: Verminderung des Y-Registers um eins
EOR: Speicher durch "EXOR" mit Accumulator verknuepfen
INC: Erhoehung des Speicherinhalts um eins
INX: Erhoehung des X-Registers um eins
INY: Erhoehung des Y-Registers um eins
JMP: Sprung zu Adresse
JSR: Unterprogrammsprung zu Adresse
LDA: Accumulator mit Wert laden
LDX: X-Register mit Wert laden
LDY: Y-Register mit Wert laden
LSR: Speicher/Accumulator um ein Bit rechtsverschieben
NOP: keine Operation
ORA: Speicher durch "OR" mit Accumulator verknuepfen
PHA: Accumulator auf Stack legen
PHP: Prozessorstatus auf Stack legen
PLA: Accumulator vom Stack holen
PLP: Prozessorstatus vom Stack holen
ROL: Linksrotation um ein Bit (Speicher, Accumulator)
ROR: Rechtsrotation um ein Bit (Speicher, Accumulator)
RTI: Ruecksprung aus Interruptsequenz
RTS: Ruecksprung aus Unterprogramm
SBC: Speicher mit Carry von Accumulator subtrahieren
SEC: Carry setzen
SED: Dezimalmodus einschalten
SEI: Interrupt sperren
STA: Accumulator in Speicher ablegen
STX: X-Register in Speicher ablegen
STY: Y-Register in Speicher ablegen
TAX: Accumulator in X-Register uebertragen
TAY: Accumulator in Y-Register uebertragen
TSX: Stackpointer in X-Register uebertragen
TXA: X-Register in Accumulator uebertragen
TXS: X-Register in Stackpointer uebertragen
TYA: Y-Register in Accumulator uebertragen

BEFEHLSLISTE

Mnemonic	ausgefuehrte Operation, symbolische Darstellung	N	V	B	D	I	Z	C
ADC	A + M + C → A, C	*	*	.	.	.	*	*
AND	A ∧ M → A	*	.	.	.	.	*	.
ASL	C ← 76543210 ← 0	*	.	.	.	.	*	*
BCC	branch on C = 0	.	.	.	.	.	.	.
BCS	branch on C = 1	.	.	.	.	.	.	.
BEQ	branch on Z = 1	.	.	.	.	.	.	.
BIT	A ∧ M, M7 → N, M6 → V	M7	M6	.	.	.	*	.
BMI	branch on N = 1	.	.	.	.	.	.	.
BNE	branch on Z = 0	.	.	.	.	.	.	.
BPL	branch on N = 0	.	.	.	.	.	.	.
BRK	PC + 2 ↓, P ↓	.	.	1	.	1	.	.
BVC	branch on V = 0	.	.	.	.	.	.	.
BVS	branch on V = 1	.	.	.	.	.	.	.
CLC	0 → C	.	.	.	.	.	.	0
CLD	0 → D	.	.	.	0	.	.	.
CLI	0 → I	.	.	.	.	0	.	.
CLV	0 → V	.	0	.	.	.	.	.
CMP	A → M	*	.	.	.	.	*	*
CPX	X → M	*	.	.	.	.	*	*
CPY	Y → M	*	.	.	.	.	*	*
DEC	M - 1 → M	*	.	.	.	.	*	.
DEX	X - 1 → X	*	.	.	.	.	*	.
DEY	Y - 1 → Y	*	.	.	.	.	*	.
EOR	A ⊻ M → A	*	.	.	.	.	*	.
INC	M + 1 → M	*	.	.	.	.	*	.
INX	X + 1 → X	*	.	.	.	.	*	.
INY	Y + 1 → Y	*	.	.	.	.	*	.
JMP	(PC + 1) → PCL (PC + 2) → PCH	.	.	.	.	.	.	.
JSR	PC + 2 ↓, (PC + 1) → PCL (PC + 2) → PCH	.	.	.	.	.	.	.
LDA	M → A	*	.	.	.	.	*	.
LDX	M → X	*	.	.	.	.	*	.
LDY	M → Y	*	.	.	.	.	*	.
LSR	0 → 76543210 → C	0	.	.	.	.	*	*
NOP	no operation	.	.	.	.	.	.	.
ORA	A ∨ M → A	*	.	.	.	.	*	.
PHA	A ↓	.	.	.	.	.	.	.
PHP	P ↓	.	.	.	.	.	.	.
PLA	A ↑	*	.	.	.	.	*	.
PLP	P ↑	vom Stack						
ROL	C ← 76543210 ← C	*	.	.	.	.	*	*
ROR	C → 76543210 → C	*	.	.	.	.	*	*
RTI	P ↑, PC ↑	vom Stack						
RTS	PC ↑, PC + 1 → PC	.	.	.	.	.	.	.
SBC	A - M - B → A	*	*	.	.	.	*	*
SEC	1 → C	.	.	.	.	.	.	1
SED	1 → D	.	.	.	1	.	.	.
SEI	1 → I	.	.	.	.	1	.	.
STA	A → M	.	.	.	.	.	.	.
STX	X → M	.	.	.	.	.	.	.
STY	Y → M	.	.	.	.	.	.	.
TAX	A → X	*	.	.	.	.	*	.
TAY	A → Y	*	.	.	.	.	*	.
TSX	S → X	*	.	.	.	.	*	.
TXA	X → A	*	.	.	.	.	*	.
TXS	X → S	.	.	.	.	.	.	.
TYA	Y → A	*	.	.	.	.	*	.

Zeichenerklaerungen

Zeichen	Erklaerung
A	Accumulator
X	X-Register
Y	Y-Register
M	Memory, Argument
P	Prozessor-status-register
S	Stack-pointer
PC	Program-counter, Programm-zaehler
PCL	PC low
PCH	PC high
0	Bitwert 0
1	Bitwert 1
→	Transfer nach
←	Transfer nach
↓	Ablegen auf Stack
↑	Holen vom Stack
+	Addition
-	Subtraktion
∧	logisches UND
∨	logisches ODER
⊻	logisches EXKLUSIV-ODER, Antivalenz
*	beeinflusst
.	keine Aenderung
=	gleich
N	Negativflag
V	Overflow, Vorzeichen-ueberlauf
B	Flag fuer Software-IRQ (BRK)
D	Flag fuer Dezimal-modus
I	Interrupt-disablebit
Z	Zeroflag, Nullflag
C	Carry, Uebertrag
B	Borrow, 1 - C, $\overline{C}$

ADRESSIERUNGSARTEN

	accumulator	immediate	zeropage	zeropage, x	zeropage, y	absolute	absolute, x	absolute, y	implied	relative	(indirect,x)	(indirect),y	(indirect)
ADC		105	101	117		109	125	121			97	113	
AND		41	37	53		45	61	57			33	49	
ASL	10		6	22		14	30						
BCC										144			
BCS										176			
BEQ										240			
BIT			36			44							
BMI										48			
BNE										208			
BPL										16			
BRK									0				
BVC										80			
BVS										112			
CLC									24				
CLD									216				
CLI									88				
CLV									184				
CMP		201	197	213		205	221	217			193	209	
CPX		224	228			236							
CPY		192	196			204							
DEC			198	214		206	222						
DEX									202				
DEY									136				
EOR		73	69	85		77	93	89			65	81	
INC			230	246		238	254						
INX									232				
INY									200				
JMP						76							108
JSR						32							
LDA		169	165	181		173	189	185			161	177	
LDX		162	166		182	174		190					
LDY		160	164	180		172	188						
LSR	74		70	86		78	94						
NOP									234				
ORA		9	5	21		13	29	25			1	17	
PHA									72				
PHP									8				
PLA									104				
PLP									40				
ROL	42		38	54		46	62						
ROR	106		102	118		110	126						
RTI									64				
RTS									96				
SBC		233	229	245		237	253	249			225	241	
SEC									56				
SED									248				
SEI									120				
STA			133	149		141	157	153			129	145	
STX			134		150	142							
STY			132	148		140							
TAX									170				
TAY									168				
TSX									186				
TXA									138				
TXS									154				
TYA									152				

DER VIDEO INTERFACE CHIP

Saemtliche Graphikmoeglichkeiten des Commodore 64, sei es nun die hochaufloesende Graphik, seien es die Sprites oder die "gewoehnliche" Darstellung von 40 Spalten mal 25 Zeilen, lassen sich auf den 6567 Video Interface Chip (auch VIC-II-Chip genannt) zurueckfuehren. Dieser sehr intelligente Baustein ermoeglicht, es beim Commodore 64 graphische Effekte in Programme einzubauen, die auf anderen Computern eines sehr grossen Programmieraufwandes beduerfen oder schlichtweg unmoeglich sind. Im folgenden soll nun auf die einzelnen Eigenschaften des VIC-II-Chips eingegangen werden, so dass es Ihnen moeglich sein wird, seine Moeglichkeiten vollstaendig auszuschoepfen, was jedoch gar nicht so einfach ist.

Zunaechst einen allgemeinen Hinweis auf die Struktur des VIC-II-Chips, die bei saemtlichen Anwendungen beruecksichtigt werden muss. Die 6567 kann, von dem vom Computer adressierbaren Bereich von 64 KB, jeweils nur 16 KB nutzen. Im Normalfall ist dies der Bereich von Adresse 0 bis 16383, der beim Einschalten des Computers als fuer den VIC-II-Chips sichtbar deklariert wird. Dieser Bereich von "nur" 16 KB resultiert aus der Anzahl an Adressleitungen, von denen der VIC-II-Chip nur 14 (anstelle der 16 Adressleitungen des Prozessors) hat. Soll nun ein anderer Bereich spezifiziert werden (dies kann immer nur in 16-KB-Bloecken geschehen), so muss dies in den Bits 0 und 1 des Ports A (Adresse 56576) der NMI-CIA (CIA #2) festgelegt werden.

Es stehen folgende Moeglichkeiten fuer den Adressbereich des VIC-II-Chips zur Verfuegung (das Datenrichtungsregister muss natuerlich auf Ausgang geschaltet sein):

Adressbereich	Bitmuster	dezimal	
0 - 16383	11	3	Normalwert
16384 - 32767	10	2	
32768 - 49151	01	1	
49152 - 65535	00	0	

Um nun auf einen anderen Bereich umzuschalten, muessen Bit 0 und Bit 1 des Port A auf den unter "dezimal" stehenden Wert gesetzt werden. Dies koennte folgendermassen vor sich gehen:

```
POKE 56576, PEEK (56576) AND 252 OR dezimal
```

Dieses 16-KB-Konzept ist Teil aller Speicherzugriffe des VIC-II-Chips und sollte daher unbedingt beachtet werden. Ist zum Beispiel der Bereich von 32768 bis 49151 gewaehlt, so liegt der Bildschirmspeicher (sofern nicht geaendert) ab der Adresse 32768 (Video-Bank-Startadresse) plus 1024 (normale Startadresse des Bildschirmspeichers) = 33792. Aber auch die Daten der Sprites und der hochaufloesenden Graphik unterliegen diesem Konzept.

Ein weiterer Hinweis bezueglich des Video-Chips: Er umfasst 47 Register, durch die dessen Funktionen gesteuert werden. Diese Register liegen im Bereich von 53248 bis 53294 und koennen mittels POKE und PEEK bearbeitet werden. Eine Uebersicht bezueglich des Video-Chips ist dem Anhang zu entnehmen.

Der Bildschirmspeicher

Der Bereich, in dem die Daten fuer die normale Darstellung von 40 mal 25 Zeichen liegen, wird Bildschirmspeicher ge-

nannt. Dies ist ein Bereich von 1000 Bytes mit einer Breite von 8 Bits. Jede Speicherzelle kann daher Werte aus dem Bereich von 0 bis 255 annehmen. Diese Werte geben im "Standard Character Mode" das Zeichen an, das an der entsprechenden Stelle des Bildschirms erscheinen soll. In anderen Darstellungsmodi kann dieser Bereich auch die Farbcodes fuer ein Feld von 8 mal 8 Dots enthalten. Darauf wird dann aber gesondert eingegangen. Der Bildschirmspeicher kann in 1-KB-Bloecken verschoben werden. Bei einem fuer den VIC-II-Chip adressierbaren Bereich von 16 KB ergeben sich daher 16 Moeglichkeiten fuer die Lage des Bildschirmspeichers. Bits 4 bis 7 von Register 24 (Adresse 53272) geben diese Startadresse an. Folgende Tabelle zeigt die moeglichen Bereiche fuer den Bildschirmspeicher:

Adressbereich	Bitmuster	dezimal	
0 - 999	0000	0	
1024 - 2023	0001	1	Normalwert
2048 - 3047	0010	2	
3072 - 4071	0011	3	
4096 - 5095	0100	4	
5120 - 6119	0101	5	
6144 - 7143	0110	6	
7168 - 8167	0111	7	
8192 - 9191	1000	8	
9216 - 10215	1001	9	
10240 - 11239	1010	10	
11264 - 12263	1011	11	
12288 - 13287	1100	12	
13312 - 14311	1101	13	
14336 - 15335	1110	14	
15360 - 16359	1111	15	

Auch hier duerfen nur die entsprechenden Bits geaendert werden, da ein Register oft mehrere Bedeutungen hat. Der Befehl zum Aendern des Adressbereichs des Bildschirms lautet ...

```
POKE 53272, PEEK (53272) AND 15 OR 16 * dezimal
```

Bei der Festlegung der Startadresse fuer die Videomatrix ist zu beachten, dass hierzu, wie bei allen Adressfestlegungen fuer den VIC-II-Chip, die Startadresse der momentanen Video-Bank zu addieren ist, um die wirkliche Startadresse des Bildschirmbereichs zu erhalten, ab der dann die Daten abgelegt werden koennen.

Dieser Bereich des Bildschirmspeichers wird nun aber noch nicht durch die Zeichenausgaberoutine des Betriebssystems unterstuetzt, da das OS (Operating System) nicht ueberprueft, ob die Startadresse des Bildschirms vom Wert 1024 abweicht. Hierzu ist es noetig, die Speicherstelle 648 zu aendern. Diese gibt die Startpage des Bildschirmspeichers an.

Dazu ein Beispiel: Wurde als Video-Bank der Bereich von 32768 bis 49151 spezifiziert und soll die Videomatrix ab Adresse 35840 liegen (dies entspricht der Adresse 3072 innerhalb des 16-KB-Bereichs), so muessen folgende Befehle eingegeben werden, damit ausserdem die Zeichenausgaberoutine des Betriebssystems die Zeichen in diesem Bereich ausgibt:

```
POKE 56576, PEEK (56576) AND 252 OR 1
POKE 53272, PEEK (53272) AND  15 OR 16 * 3
POKE   648, 35840 / 256
```

Hierdurch ist es zum Beispiel auch moeglich, den Bildschirmanfang auf die Adresse 32768 zu legen, da sich auf diese Weise das Umschreiben von Programmen fuer die CBMs mit 40 Zeichen je Zeile wesentlich vereinfacht. Lediglich der Bereich fuer die Farben der einzelnen Zeichen muss hierbei noch miteinbezogen werden.

Da der Bildschirmbereich nur 1000 Bytes umfasst, bleiben am Ende noch 24 Bytes des 1-KB-Blocks fuer die Videomatrix uebrig. Hiervon sind die ersten 16 Bytes unbenutzt, die letzten acht Bytes stehen im Zusammenhang mit den Sprites und werden bei der Erklaerung der Sprites ausfuehrlich erlaeutert.

Der Farbspeicher

Um jedes einzelne Zeichen des Bildschirmspeichers in einer der 16 verschiedenen Farben darzustellen, wird der Farbspeicher benoetigt. Dieser RAM-Bereich umfasst 1024 Adressen zu je 4 Bits (Bit 0 bis Bit 3), wobei jede Zelle einen Wert von 0 bis 15 enthaelt, der fuer die jeweilige Farbe steht. Der Farbspeicher hat jedoch auch noch andere Aufgaben, auf die dann aber im entsprechenden Zusammenhang naeher eingegangen werden wird.

Der Farbspeicher kann nicht in einen anderen Bereich verlegt werden und liegt daher konstant an den Adressen 55296 bis 56319, wobei jedoch auch hier, wie bei der Videomatrix, nur die ersten 1000 Nybbles (damit ist ein Block von vier Bits gemeint) benutzt sind. Die restlichen 24 Nybbles sind unbenutzt.

Der Zeichengenerator (Definition eigener Zeichen)

Im Zeichengenerator sind die Informationen gespeichert, wie die Zeichen, die im normalen Darstellungsmodus auf dem Bildschirm erscheinen, aussehen. Da ein Zeichen eine Gebiet von 8 mal 8 Punkten umfasst, werden zur Definition eines Zeichens 64 Bits (entsprechend 8 Bytes) benoetigt. Bei einem Zeichenvorrat von 256 Zeichen ergibt dies 256 * 8 Bytes gleich 2 KB, die ein kompletter Zeichensatz einnimmt.

Die Lage des Zeichensatzes ist in 2-KB-Bloecken frei waehlbar, daraus resultieren 8 Moeglichkeiten fuer den Bereich des Zeichengenerators. Die Bits 1 bis 3 des Registers 24 (Adresse 53272) muessen zur Festsetzung der Startadresse geaendert werden. Der Zeichengenerator-ROM des Betriebssystems liegt in den Adressen von 53248 bis 57343. Es werden 4 KB benoetigt, da der Commodore 64 schliesslich zwei vollstaendige Zeichensaetze vorsieht und bei Umschaltung von einem zum anderen Zeichensatz nur die Startadresse geaendert wird. Dieser 4-KB-Bereich ist im Normalfall jedoch fuer I/O und das Farb-RAM vorgesehen, und der Benutzer kann nicht direkt auf die Daten des Zeichengenerators zugreifen.

Das Einblenden des Zeichengenerators und das Ausblenden der I/O-Bausteine kann durch Loeschen von Bit 2 im Prozessorport der CPU 6510 erfolgen. Allerdings muss zuvor der Interrupt abgeschaltet werden (Bit 0 des Adresseninhalts von 56334 loeschen), da das Betriebssystem auf diesen Bereich zugreift und, sofern die I/O-Register nicht erreichbar sind, den Computer abstuerzen laesst. Nach Zugriff auf den Zeichengenerator sollte dieser durch Setzen von Bit 2 des Prozessorports wieder ausgeblendet werden und das Interrupthandling durch Setzen von Bit 0 des Kontrollregisters A wieder aufgenommen werden. Um zum Beispiel den Zeichengenerator aus dem Bereich von 53248 bis 55295 in den Bereich von 14336 bis 16383 zu

uebertragen kann folgendermassen vorgegangen werden:

```
POKE 56334, PEEK(56334) AND 254
POKE 1, PEEK(1) AND 251
FOR I=53248 TO 55295 : POKE I-38912, PEEK(I) : NEXT
POKE 1, PEEK(1) OR 4
POKE 56334, PEEK(56334) OR 1
```

Da der VIC-II-Chip jedoch immer nur auf einen 16-KB-Bereich zugreifen kann, muss das Zeichengenerator-ROM fuer den Video-Chip auf eine weitere Weise verfuegbar sein. Schliesslich liegen die Adressen von 53248 bis 57343 nicht im Bereich der Standard Video-Bank von 0 bis 16383. Daher ist der Zeichengenerator zusaetzlich in den Bereichen von 4096 bis 8191 sowie 36864 bis 40959 sichtbar, dies jedoch nur (!) fuer den VIC-II-Chip. RAM, das in diesem Bereich liegt, ist daher ohne jegliche Einschraenkungen benutzbar. Auch wenn der Zeichengenerator in den vom Prozessor lesbaren Bereich eingeblendet wird, so bleiben die obengenannten Bereiche davon unberuehrt. Soll jedoch der VIC-II-Chip auf einen RAM-Bereich zugreifen, der auch von Character-ROM "unterlegt" ist, so wird dieser sich fuer den ROM-Bereich entscheiden. Eigene Zeichensaetze, Bildschirmdaten, HIRES-Graphiken etc. sollten daher in anderen Bereichen abgelegt werden.

Hier eine Uebersicht ueber die moeglichen Bereiche des Zeichengenerators. Zu den Bereichsadressen ist auch hier die Bankstartadresse zu addieren:

Adressbereich	Bitmuster	dezimal	
0 - 2047	000	0	
2048 - 4095	001	1	
4096 - 6143	010	2	Normalwert
6144 - 8191	011	3	
8192 - 10239	100	4	
10240 - 12287	101	5	
12288 - 14335	110	6	
14336 - 16383	111	7	

In Video-Bank 0 und 2 sind die Bereiche 4096 bis 8191 von der Kopie des Zeichengenerators belegt. Zum Aendern der Startadresse des Zeichengenerators dient der folgende Befehl:

```
POKE 53272, PEEK (53272) AND 241 OR 2 * dezimal
```

Um nun ein eigenes Zeichen zu definieren muss folgendermassen vorgegangen werden: Der Bildschirmcode des zu definierenden Zeichens wird mit 8 multipliziert (jedes Zeichen benoetigt zur Darstellung 8 Bytes) und zur Startadresse des Zeichengenerators addiert. Diese Adresse gibt nun die Startadresse fuer die Daten dieses speziellen Zeichens an.

Angenommen, folgendes Summenzeichen solle definiert werden:

```
Zeichen    Binaer     Dezimal

 ******    oIIIIIIo     126
 **  **    oIIooIIo     102
  **       ooIIoooo      48
   **      oooIIooo      24
  **       ooIIoooo      48
 **  **    oIIooIIo     102
 ******    oIIIIIIo     126
           oooooooo       0
```

Hierzu gleich ein Hinweis: Es sollten immer mindestens zwei gesetzte Punkte nebeneinander liegen, da es ansonsten zu farblichen Veraenderungen in der Darstellung kommen kann.

Ein gesetztes Bit entspricht nun einem Punkt in der Farbe, die im zugehoerigen Nybble des Farb-RAMs spezifiziert wurde. Geloeschte Bits werden als Dots in der momentanen Hintergrundfarbe dargestellt.

Um nun dieses Zeichen in den Zeichensatz, den wir zuvor in den Bereich von 14336 bis 16383 uebertragen haben, einzubinden, muessen die unter der Spalte 'Dezimal' stehenden Zahlen in die Zeichengeneratortabelle uebertragen werden. Vergessen Sie nicht, den Bereich ab 14336 vor Zugriff durch BASIC mittels den Befehlen ...

```
POKE 55, 0 : POKE 56, 56 : CLR
```

... zu schuetzen, da sonst der Zeichengenerator ueberschrieben werden kann (die Befehlsfolge sollte in der ersten Zeile des Programms stehen). Soll das Sigma-Zeichen nun dem Bildschirmcode 28 (im Normalfall das Pfund-Zeichen) zugeordnet werden, so muessen die Daten fuer das neue Zeichen ab der Position 14336 (Startadresse des Zeichengenerators) plus 8 * 28 (Zeichencode) abgelegt werden. Dies kann zum Beispiel erfolgen durch ...

```
FOR I=14560 TO 14567 : READ A : POKE I, A : NEXT
DATA 126, 102, 48, 24, 48, 102, 126, 0
```

Auch der Code Null ist in diesem Fall wichtig, da schliesslich alle 8 Zeilen des Zeichens neu belegt werden muessen. Nun muss der Zeichensatz mit dem neuen Zeichen noch aktiviert werden. Dies erfolgt durch den Befehl zur Festsetzung der Startadresse eines Zeichengenerators, in diesem Spezialfall fuer die Startadresse 14336 durch ...

```
POKE 53272, PEEK (53272) AND 241 OR 2 * 7
```

Sollten Sie die auf dieser und der vorherigen Seite aufgefuehrten Befehlskombinationen in der korrekten Reihenfolge (oberen Bereich vor BASIC schuetzen, Zeichensatz kopieren, neues Zeichen in Zeichensatz uebertragen, neuen Zeichengenerator aktivieren) eingegeben haben, so sollte jetzt jedesmal, wenn Sie die Pfund-Taste druecken, das Summenzeichen erscheinen.

Hierzu noch Anmerkungen: Wird nun mittels der Commodore-Taste oder ueber CHR$ in einen anderen Zeichensatz umgeschaltet, so wird der Bildschirm keinerlei Zeichen mehr enthalten (meist wohl irgendwelche Punkte und Linien), da durch obige Befehlsfolgen nur ein Zeichensatz definiert wurde. Es ist jedoch moeglich, einen zweiten Zeichensatz zu definieren, der dann auch (wie der zweite interne Zeichensatz) direkt angesprochen werden kann. Hierbei muessen jedoch beide Zeichensaetze innerhalb eines 4-KB-Blocks liegen.

Ein weiterer Punkt: Auch wenn nun das Zeichen definiert wurde, so ist das Pfund-Zeichen noch immer als negatives Zeichen vorhanden. Dies ist erkennbar, wenn man zum Beispiel den Cursor auf ein Sigma-Zeichen bewegt. Negative Zeichen muessen daher, sofern erwuenscht, getrennt definiert werden. Soll das bereits als positives Zeichen vorhandene Sigma auch als negatives Zeichen mit dem Code 156 (128 plus 28) erreichbar sein, so kann die Definition durch ...

```
FOR I = 14336 + 156 * 8 TO 14336 + 156 * 8 + 7
READ A : POKE I, 255 - A : NEXT
```

... erfolgen (vergessen Sie den RESTORE-Befehl nicht, falls Sie die gleichen Daten ein weiteres Mal verwenden). Bei negativen Zeichen werden gesetzte Bits durch geloeschte Bits ersetzt und umgekehrt.

Durch die Zeichendefinition ist es moeglich, den gesamten Zeichensatz umzudefinieren. Allerdings bewirkt ein veraendertes Aussehen an den Zeichen nichts. Sie haben noch immer die gleichen Funktionen, die sie auch vorher hatten, auch wenn sie in BASIC-Listings vielleicht ein wenig seltsam aussehen moegen.

Hier noch einmal kurz die Vorgehensweise, die 8 Bytes an Daten fuer den Zeichensatz aus einem fertigen 8 mal 8 Punktemuster zu erzeugen: Jede der acht Zeilen wird getrennt bearbeitet und in der Reihenfolge von oben nach unten in der DATA-Zeile abgelegt. Die acht Punkte innerhalb einer Zeile werden als Binaerzahl aufgefasst. Ein gesetzter Punkt entspricht daher der Ziffer 1, ein geloeschter Punkt dem Ziffernwert 0. Jede Stelle innerhalb des Bytes (8 Bits) hat nun einen bestimmten Wert. Dies sind die Potenzen von 2. Das linke Bit hat den Wert 128 (entsprechend 2 hoch 7), das rechte Bit den Wert 1 (entsprechend 2 hoch 0). Jeder Wert einer Bitposition wird nun mit dem Zifferwert dieses Bits multipliziert (das Ergebnis ist also entweder gleich null oder gleich dem Bitwert) und alle Ergebnisse addiert. Dies ist dann einer der acht Werte.

Modus fuer erweiterte Hintergrundfarben

Oft ist es wuenschenswert, mehr als nur eine Hintergrundfarbe fuer ein Zeichen zur Verfuegung zu haben. Fuer solche Anwendungen ist der Modus fuer erweiterte Hintergrundfarben (Extended Background Color Mode) gedacht, der es ermoeglicht, jedem Zeichen eine der vier globalen Hintergrundfarben zuzuordnen. Ausserdem ist es weiterhin moeglich, eine der sechzehn Vordergrundfarben fuer das Zeichen auszuwaehlen. Das Aktivieren dieses Modus' erfolgt durch Setzen von Bit 6 in Register 17 (Adresse 53265) des VIC-II-Chips, wodurch Bit 6 und Bit 7 des in der Videomatrix stehenden Zeichencodes nun als Information ueber die Hintergrundfarbe des Zeichens verarbeitet werden. Aus der Kombination von zwei Bits ergeben sich nun folgende vier Moeglichkeiten:

Bit 7	Bit 6	Dezimalbereich	Hintergrundfarbregister
0	0	0 - 63	Background Color #0 (53281)
0	1	64 - 127	Background Color #1 (53282)
1	0	128 - 191	Background Color #2 (53283)
1	1	192 - 255	Background Color #3 (53284)

Aus der Verwendung der Bits 6 und 7 als Hintergrundfarbpointer (schliesslich geben die beiden Bits nicht die Farbe selbst, sondern das Background Color Register, aus dem die Farbe stammt, an) resultiert, dass nur noch die Zeichencodes von 0 bis 63 verwendet werden koennen. Alle Zeichen mit einem Bildschirmcode ab 64 werden wieder in den Bereich von 0 bis 63 umgewandelt und mit der zugehoerigen Hintergrundfarbe dargestellt.

In BASIC-Programmen lassen sich die Bildschirmcodes recht gut erreichen. Die Zeichen von 0 bis 63 (entsprechend den CHR$-Codes von 32 bis 95) werden wie gewohnt, dass heisst in

der "normalen" Hintergrundfarbe, dargestellt. Die gleichen Zeichen mit den Codes von 64 bis 127 (also mit der Hintergrundfarbe #1, die in Adresse 53282 festgelegt wird) koennen zum groessten Teil (alle Buchstaben, die restlichen Zeichen muessen leider gesucht werden, da nicht bei jeder Taste die Kombination mit der Shift-Taste eine Erhoehung des Zeichencodes um 64 bewirkt) durch zusaetzliches Druecken der Shift-Taste erreicht werden. Negative Zeichen wiederum werden in den Hintergrundfarben #2 und #3 dargestellt.

Da die Farben selbst nicht dem Zeichencode direkt mitgegeben werden (sondern nur die Information, wo die Farbe zu finden ist), ist es zum Beispiel moeglich, saemtlichen Zeichen einer Hintergrundfarbe gleichzeitig eine neue Hintergrundfarbe zuzuordnen, ohne dass etwas am Zeichencode geaendert werden muss (es muss einfach nur das Register fuer die entsprechende Hintergrundfarbe geaendert werden). Die Vordergrundfarbe, also die Farbe, in der das Zeichen selbst dargestellt wird, kann (wie im normalen Darstellungsmodus auch) ueber die Farbtasten geandert werden und wird weiterhin im Color Memory (Farb-RAM) im Adressbereich von 55296 bis 56295 abgelegt.

Mehrfarbige Zeichen (Multicolor Modus)

Oft ist es erwuenscht, ein Zeichen innerhalb eines 8 mal 8 Feldes in mehreren Farben darzustellen. Hierzu eignet sich moeglicherweise der 'Modus fuer erweiterte Hintergrundfarben', der mit dem hier beschriebenen Modus NICHT kombinierbar ist. Eine weitere Moeglichkeit fuer eine groessere Farbauswahl ist der Multicolor Modus. Dieser wird durch Setzen von Bit 4 im Register 22 (Adresse 53270) eingeschaltet. Dadurch werden die Daten aus dem Zeichengenerator ein wenig anders interpretiert, als dies der Fall ist, wenn dieses Bit geloescht ist.

Voraussetzung fuer die Darstellung eines Zeichens im Multicolor Modus ist, dass Bit 3 im Color-Nybble-RAM gesetzt ist (also eine Farbe ueber die Commodore Taste ausgewaehlt wurde). Ist dies nicht der Fall, so wird das Zeichen in der gewohnten Weise dargestellt (allerdings stehen fuer diese 'gewohnte Darstellungsweise' - durch die Benutzung von Bit 3 als Flag - nur noch die ersten acht Farben zur Verfuegung). Eine Mischung beider Modi ist daher problemlos moeglich. Bei gesetzem Bit 3 im Farb-RAM werden jeweils zwei Bits aus dem Zeichengenerator fuer die Auswahl der Farbe verwendet. Da aber auch jeweils zwei Punkte angesprochen werden, halbiert sich die effektive Aufloesung auf eine Groesse von 4 (horizontal) mal 8 (vertikal) doppelten Dots. Die Groesse des Zeichens bleibt daher bei 8 mal 8 Punkten. Die einzelnen Bitpaare koennen folgende Zustaende annehmen und haben daher folgende Bedeutungen:

Bitpaar	Farbquelle	Funktion
00	Background #0 (53281)	Hintergrund
01	Background #1 (53282)	Hintergrund
10	Background #2 (53283)	Vordergrund
11	Color RAM, Bits 0 - 2	Vordergrund

Auch fuer die zum Bitpaar '11' gehoerigen Dots stehen nur noch die ersten acht Farben zur Verfuegung, da nur noch Bit 0 bis Bit 2 des Farb-RAM zur Verwendung freistehen. Die Spalte 'Funktion' haengt mit der Benutzung von Sprites zusammen und hat ansonsten keinerlei Bedeutung.

Hierzu ein Beispiel: Die Voraussetzungen fuer dieses Beispiel sind identisch mit denen fuer die Definition des Sigma-Zeichens, allerdings muss zusaetzlich das Multicolor Bit gesetzt und die DATA-Zeile geaendert werden.

Es soll folgendes Zeichen (bestehend aus drei Farben (+*#) plus der fuer den Hintergrund zustaendigen Farbe) definiert werden:

Zeichen	Farbgruppen	Binaer	Dezimal
++####**	01 11 11 10	oIIIIIIo	126
++ **	01 00 00 10	oIooooIo	66
++**	00 01 10 00	oooIIooo	24
++**	00 01 10 00	oooIIooo	24
**++	00 10 01 00	ooIooIoo	36
**++	00 10 01 00	ooIooIoo	36
** ++	10 00 00 01	IooooooI	129
**####++	10 11 11 01	IoIIIIoI	190

Die Datenzeile muss nun natuerlich lauten ...

```
DATA 126, 66, 24, 24, 36, 36, 129, 190
```

Das Einschalten des Multicolor Modus erfolgt durch ...

```
POKE 53270, PEEK (53270) OR 16
```

Bei vorheriger Auswahl einer Farbe durch die Commodore-Taste (!) und Druecken der Pfund-Taste wird nun ein Zeichen ausgegeben, das aus zwei Diagonalen und zwei am oberen und unteren Ende des Zeichens befindlichen Balken (in der spezifizierten Farbe) besteht. Die Farben der Diagonalen haengen von den Registern 53282 und 53283 ab, von denen das erstere die Farbe der Diagonalen von links oben nach rechts unten angibt und das zweite die der anderen Diagonalen. Allerdings, und das sollte unbedingt erwaehnt werden, ist die Farbe nicht unbedingt so sauber, wie dies bei grossen Flaechen (Rahmen oder Hintergrund) der Fall ist. Sie haengt in erster Linie vom verwendeten Fernseher beziehungsweise Monitor ab. Durch POKE-Befehle lassen sich nun die Farben der Diagonalen veraendern. Hier stehen jedoch, im Gegensatz zum Bitpaar '11', alle 16 Farben frei zur Verfuegung.

Wird das definierte Zeichen allerdings mit einem Cursorfarbcode von 0 bis 7 (oder bei geloeschtem Multicolor Bit) ausgegeben, so wird das Aussehen des Zeichens sich aendern, da hier, wie dies auch beim Sigma der Fall war, dann nur noch nach gesetzten und geloeschten Bits mit einer Aufloesung von 8 mal 8 Punkten unterschieden wird. Andererseits ergeben sich recht interressante Effekte, wenn bei eingeschaltetem Multicolor Modus mit normalen Zeichen gearbeitet wird, die im Mehrfarbenmodus dargestellt sind.

Auch hier ist es, wie dies in aehnlicher Form im Modus fuer erweiterte Hintergrundfarben moeglich war, durch Aendern eines Registers moeglich, saemtliche zugehoerige Bitpaare (mit Ausnahme des Bitpaars 11) auf einmal zu aendern. Diesem Prinzip entspricht uebrigens auch die Aenderung der Hintergrundfarbe im normalen Darstellungsmodus. Hier werden saemtliche Bits mit dem Wert 0 (die man als 'nicht sichtbar' bezeichnen wuerde, da sie der Hintergrund des Zeichens sind) in einer globalen Farbe dargestellt, im Gegensatz zu den gesetzten Bits, die die zugehoerigen Dots in der im Color Nybble RAM spezifizierten Farbe erscheinen lassen. So sind im Multicolor Modus drei globale Farben vorhanden.

Die bis hier beschriebenen Darstellungsmodi, die sich alle unter dem Begriff "Zeichendarstellung" zusammenfassen lassen, sind alle Moeglichkeiten der Darstellung bei Zugriff auf den Zeichengenerator.

Die hochaufloesende Graphik (HIRES, High Resolution)

Zusaetzlich zu den vorher beschriebenen Darstellungsmodi, die in irgendeiner Form auf einen Zeichengenerator zugreifen, existiert ein weiterer Darstellungsmodus, in dem jeder der 64000 Bildpunkte getrennt angesprochen werden kann. Dieser Modus wird daher 'hochaufloesend' genannt. Er eignet sich speziell fuer die Darstellung von mathematischen Funktionen, die mit grosser Genauigkeit aufgeloest werden koennen, oder als Hintergrund fuer Spiele, da Sprites auch mit HIRES-Graphiken kombinierbar sind.

Eine Programmierung dieses Bit Map Modus (jedes Bit aus dem fuer die hochaufloesende Graphik spezifizierten Bereich ist fuer einen Bildschirmpunkt zustaendig) ist sehr aufwendig. BASIC ist hierzu speziell aus Geschwindigkeitsgruenden nur sehr bedingt geeignet. Man wird bei groesseren Programmen auf die Verwendung von Maschinenprogrammen zum Setzen und Loeschen von Punkten, Linien und Flaechen wohl nicht verzichten koennen, falls nicht gar das gesamte Programm in Maschinensprache geschrieben werden muss.

Fuer die hochaufloesende Graphik wird ein Bereich von 8000 Bytes an Speicher benoetigt, der im Normalfall von BASIC-Speicher abgezweigt werden muss. Diese 8 KB werden direkt auf dem Bildschirm dargestellt. Es ist daher moeglich, den Zustand (genaugenommen die Farbe) eines jeden Punktes direkt zu aendern. Wie auch bei der Darstellung von Zeichen aus dem Zeichengenerator, so existieren auch im Bit Map Modus unterschiedliche Formen. Dies ist der Standard Bit Map Mode und der Multicolor Bit Map Mode. Die horizontale Aufloesung wird, wie dies bei mehrfarbigen Zeichen auch der Fall war, im Multicolor Bit Map Modus auf die Haelfte (160 doppelte Dots) reduziert, da auch hier immer zwei Dots gleichzeitig angesprochen werden. In beiden Modi bekommt ausserdem die Videomatrix eine neue Bedeutung. Durch sie werden nun zwei der moeglichen Farben innerhalb eines Feldes von 8 mal 8 Dots festgelegt.

Das Einschalten des Bit Map Modus' erfolgt durch Setzen von Bit 5 in Register 17 (Adresse 53265), die Startadresse des 8-KB-Bereichs (wodurch sich durch die 16-KB-Architektur des VIC-II-Chips zwei Moeglichkeiten ergeben) wird durch Bit 3 in Register 24 (Adresse 53272) festgelegt (natuerlich muss auch hier wieder die Startadresse der Video-Bank addiert werden):

Adressbereich	Bitwert
0 - 7999	0
8192 - 16191	1

Dieses Bit wird auch zur Festsetzung der Startadresse des Zeichengenerators benutzt, der jedoch im HIRES-Modus nicht verwendbar ist. Die Startadresse des Zeichengenerators muss daher wieder neu gesetzt werden, falls in einen Modus geschaltet werden soll, der den Zeichengenerator benoetigt.

Der Bereich von 0 bis 7999 kann in Video-Bank 0 natuerlich nicht benutzt werden, da ansonsten wichtige Systemadressen ueberschrieben werden muessten.

Standard Bit Map Modus

In diesem Modus bleibt das Color Nybble RAM unbeachtet, nur die Videomatrix und der 8-KB-Bereich fuer die 64000 Punkte werden verwendet. Das erste Byte aus dem 8-KB-Bereich ist fuer die ersten acht Punkte zustaendig. Dabei erfolgt die Unterteilung innerhalb des Bytes von Bit 7, erster Dot (oben links) bis Bit 0, achter Dot. Das zweite Byte bestimmt die ersten acht Dots innerhalb der zweiten Dotzeile. Dies geht so weiter bis zum achten Byte, das die ersten acht Dots der achten Dotzeile bestimmt. Ab dem neunten Byte wird nun wieder in die erste Zeile zurueckgegangen. Hier bestimmen die Bitzustaende nun das neunte bis sechzehnte Dot von links. Auch dieses Prinzip wird (wie man sich bei einer horizontalen Aufloesung von 320 Dots leicht ausrechnen kann) bis zum 320sten Byte fortgesetzt. Die nun folgenden 320 Bytes bestimmen die Dotzustaende der Zeilen 9 bis 16. Dieses geht so weiter, bis alle der 320 mal 200 Punkte bearbeitet sind. Allein an dieser zwar sehr logischen aber doch fuer BASIC sehr umstaendlich zu handhabenden Aufteilung erkennt man, dass wohl viel Zeit allein fuer die Berechnung der Bitposition benoetigt wird. Hier eine Darstellung fuer den Aufbau des Bildschirms aus den zugehoerigen Bytes:

	Spalte				
	0 - 7	8 - 15	...	304 - 311	312 - 319
Reihe 0:	Byte 0,	Byte 8,	...,	Byte 304,	Byte 312
Reihe 1:	Byte 1,	Byte 9,	...,	Byte 305,	Byte 313
... ...:	... ...,	... ...,	...,	... ...,	... ...
Reihe 6:	Byte 6,	Byte 14,	...,	Byte 310,	Byte 318
Reihe 7:	Byte 7,	Byte 15,	...,	Byte 311,	Byte 319
Reihe 8:	Byte 320,	Byte 328,	...,	Byte 624,	Byte 632
Reihe 9:	Byte 321,	Byte 329,	...,	Byte 625,	Byte 633
... ...:	... ...,	... ...,	...,	... ...,	... ...
Reihe 14:	Byte 326,	Byte 334	...,	Byte 630,	Byte 638
Reihe 15:	Byte 327,	Byte 335,	...,	Byte 631,	Byte 639
...					
...					
Reihe 192:	Byte 7680,	Byte 7688,	...,	Byte 7984,	Byte 7992
Reihe 193:	Byte 7681,	Byte 7689,	...,	Byte 7985,	Byte 7993
... ...:	... ...,	... ...,	...,	... ...,	... ...
Reihe 198:	Byte 7686,	Byte 7694,	...,	Byte 7990,	Byte 7998
Reihe 199:	Byte 7687,	Byte 7695,	...,	Byte 7991,	Byte 7999

Ist nun ein Bit gesetzt, so wird die Farbe des zugehoerigen Punktes vom hoeherwertigen Nybble (MSN, most significant nybble) innerhalb der Videomatrix bestimmt, bei geloeschten Bits ist dies das niederwertige Nybble (LSN, least significant nybble) der Videomatrix:

Bitwert	Farbe spezifiziert durch ...
0	LSN (Bits 0 - 3) des zugehoerigen Videomatrixbytes
1	MSN (Bits 4 - 7) des zugehoerigen Videomatrixbytes

Es stehen also zwei unterschiedliche Farben, die NICHT aus dem Color-Nybble-RAM stammen, innerhalb eines 8 mal 8 Feldes zur Verfuegung. Bei der Darstellung von Funktionen werden sich diese beiden Farben wohl ueber den gesamten Bildschirm erstrecken, wobei dann eine Farbe als Hintergrundfarbe verwendet werden wird und eine andere Farbe fuer den Funktions-

graphen selbst. Bei vier zur Verfuegung stehenden Bits fuer jede Farbe koennen daher alle sechzehn Farben verwendet werden.

Hier noch eine Umrechnungmethode in Form eines BASIC-Unterprogramms, das durch GOSUB aufgerufen werden kann. Die Variablen X und Y geben die Koordinate des zu setzenden Punkts an:

```
B = 8192 + (X AND 504) + 40 * (Y AND 248) + (Y AND 7)
POKE B, PEEK (B) OR 2 ↑ (7 - (X AND 7)) : RETURN
```

Dieses Programm geht von einem Bit Map Bereich von 8192 bis 16191 aus. Wie erkennbar, ist dieses Programm sehr leicht in Maschinensprache zu uebersetzen, da saemtliche Operationen direkt durchfuehrbar sind (teilweise jedoch 16 Bit). Die Multiplikation mit 40 muss in mehreren Schritten durchgefuehrt werden: Als erstes wird der Operand mit 4 multipliziert, dann wird der Operand selbst noch einmal addiert (dies entspricht einer Multiplikation mit 5). Das Ergebnis wird dann noch dreimal linksverschoben. Eine Potenzierung zur Basis zwei kann durch Verschiebung eines gesetzten Bits innerhalb eines Bytes erfolgen.

Zum Loeschen eines Punktes muss in der obigen Routine lediglich der POKE-Befehl etwas geaendert werden:

```
POKE B, PEEK (B) AND 255 - 2 ↑ (7 - (X AND 7))
```

Um nun zum Beispiel eine Sinuskurve auf den Bildschirm zu bringen, kann folgendes Programm verwendet werden:

```
POKE 53280, 14 : POKE 55, 0 : POKE 56, 32 : CLR
FOR I = 8192 TO 16191 : POKE I, 0 : NEXT
FOR I = 1024 TO  2023 : POKE I, 1 : NEXT
POKE 53272, PEEK (53272) OR 8
POKE 53265, PEEK (53265) OR 32
FOR X = 0 TO 319 : Y = 100 - 100 * SIN (X * pi / 160)
GOSUB punktsetzroutine : NEXT
```

Dabei ist mit 'pi' natuerlich die Shift-Funktion der Taste zwischen 'RESTORE' und '*' gemeint. Als Punktsetzroutine kann zum Beispiel die obige verwendet werden. Um die durch das Wort 'READY.' erzeugten Farbfelder zu vermeiden, kann man am Ende des Programms eine Endlosschleife anfuegen.

Der Abbruch sollte durch RUNSTOP und RESTORE erfolgen, da dann wieder in den Normalmodus zurueckgekehrt wird. Der Pointer auf das Ende des Arbeitsspeichers kann dann auch wieder rueckgesetzt werden. In diesem speziellen Programm waere es nicht einmal noetig gewesen, den Pointer (55/56) zu aendern, da keinerlei Strings verwendet werden. Die Ablage von Variablen ist jedoch bereits anderweitig erklaert.

Multicolor Bit Map Modus

Waehrend im Standard Bit Map Modus nur zwei verschiedene Farben für jeden Punkt zur Verfuegung stehen, so sind dies im Multicolor Modus vier verschiedene Farben. Mehrere Farben koennen benoetigt werden, wenn zum Beispiel zwei verschiedenfarbige Linien sich kreuzen oder einfach, um bessere Farbgraphiken zu erstellen. Wie dies beim Multicolor Modus fuer Zeichen jedoch auch der Fall war, so muss auch hier die Haelfte der horizontalen Aufloesung geopfert werden. Es stehen hier also noch 160 (horizontal) mal 200 (vertikal) doppelte Dots zur Verfuegung, da auch hier jeweils zwei Bits an

Daten im Bit Map Speicherbereich fuer zwei Dots Gueltigkeit haben.

Initialisiert wird der Multicolor Bit Map Modus durch Setzen des Multicolor Bits (Bit 4) in Register 22 (Adresse 53270) sowie natuerlich durch Setzen des Bits fuer den Bit Map Modus, Bit 5 in Register 17 (Adresse 53265).

Die Anordnung der Bytes auf dem Bildschirm bei eingeschaltetem Multicolor Modus ist identisch mit der des Standard Bit Map Modus. Lediglich die Bedeutungen der Bits innerhalb eines Bytes sind unterschiedlich. Bei zwei Bits an Daten fuer jeweils zwei Dots stehen folgende vier Moeglichkeiten zur Auswahl:

Bitpaar	Farbquelle	Funktion
00	Background #0 (53281)	Hintergrund
01	MSN (Videomatrix)	Hintergrund
10	LSN (Videomatrix)	Vordergrund
11	Color Nybble RAM	Vordergrund

Die Spalte 'Funktion' steht auch hier im Zusammenhang mit den Sprites.

Wie ersichtlich, existiert eine globale Farbe fuer den gesamten Bildschirm. Ausserdem stehen innerhalb eines jeden 8 mal 8 Feldes weitere drei Farben zur Verfuegung. Innerhalb eines 8 mal 8 Feldes koennen daher maximal vier Farben Verwendung finden. Waehrend das Farb-RAM im Standard Bit Map Mode unbenutzt war, dient es hier als Farbquelle bei der Darstellung des Bitpaars '11'.

Die SPRITES (Movable Object Blocks, MOBs)

Die Sprites sind, um es einmal ganz bescheiden auszudruekken, das absolute Nonplusultra des VIC-II-Chips. Durch sie ist es zum Beispiel moeglich, Spielprogramme so zu vereinfachen, dass selbst in BASIC noch schnelle Graphikspiele problemlos auch fuer den Einsteiger zu programmieren sind. In Maschinensprache ergibt sich dann sogar die Steigerung dieses Superlativ, da man viele Moeglichkeiten (gerade im Zusammenhang mit der hochaufloesenden Graphik) von BASIC aus gar nicht voll ausnutzen kann.

Bei den Sprites handelt es sich um eine Art ueberdimensioniertes Zeichen in einer Groesse von 24 (horizontal) mal 21 (vertikal) Bildschirmpunkten. Diese 504 Punkte eines Sprites koennen beliebig gesetzt und geloescht werden, so wie das bei der hochaufloesenden Graphik auch der Fall war. Sprites koennen ausserdem mit jedem Bildschirmmodus uneingeschraenkt kombiniert werden, ganz gleich, ob es sich dabei um die hochaufloesende Graphik, einen Multicolor Modus oder die Darstellung von einzelnen (durch den Zeichengenerator definierten) Zeichen handelt.

Die Programmierung der Sprites erfolgt unabhaengig von der des restlichen Bildschirms. So muss bei der Erstellung fuer die Graphik eines Spiels in keinster Weise auf die Bewegungen der MOBs Ruecksicht genommen werden. Allerdings ist es moeglich, Kollisionen eines Sprites mit dem uebrigen Bildschirm oder mit anderen der ingesamt acht verchiedenen Sprites zu erkennen. Es muessen daher noch nicht einmal Abfragen auf die Positionen der Sprites in ein Programm eingebaut werden, da diese Ueberpruefungen bereits durch den MOS 6567 durchgefuehrt werden.

Jedes Sprite laesst sich in X- und Y-Richtung bewegen. Man gibt dem VIC-II-Chip dazu nur die gewuenschte Position an, worauf das Sprite an diese Position gesetzt wird. Auch sind Ueberlagerungen von Sprites, Bildschirmvorder- und Hintergrund moeglich, so dass der Eindruck einer dreidimensionalen Darstellung erreicht werden kann.

Der Aufbau der Sprites

Jedes Sprite besteht aus 504 Punkten, die einzeln beeinflusst werden koennen. Ein Sprite stellt also ein kleines Gebiet hochaufloesender Graphik dar. Um diese 504 Punkte zu definieren, benoetigt man 63 Bytes, da jedes Byte aus 8 Bit (und jedes Bit ist fuer einen Dot zustaendig) besteht. Diese 63 Bytes werden in einem Block im Arbeitsspeicher abgelegt. Die Zuordnung der Bytes innerhalb des Gebiets von 24 mal 21 Punkten ist folgendermassen: Das erste Byte bestimmt die ersten acht Punkte der ersten Zeile, das zweite Byte die Punkte 9 bis 16, das dritte die Punkte 17 bis 24 der ersten Zeile. Die naechsten drei Bytes sind fuer die zweite Zeile zustaendig. Dies geht so bis zum 21. Block von drei Bytes weiter. Innerhalb eines Bytes ist (wie dies auch bei der Definition von eigenen Zeichen oder bei der hochaufloesenden Graphik der Fall war) Bit 7 dem ersten Bildschirmpunkt innerhalb der Reihe von acht Punkten zugeordnet, Bit 0 entsprechend dem letzen Dot. Zum Aufbau hier noch eine anschauliche Darstellung:

```
                     Spalte

              0 -  7   8 - 15  16 - 23

Zeile  0: Byte  0, Byte  1, Byte  2
Zeile  1: Byte  3, Byte  4, Byte  5
Zeile  2: Byte  6, Byte  7, Byte  8
Zeile  3: Byte  9, Byte 10, Byte 11
...
Zeile 17: Byte 51, Byte 52, Byte 53
Zeile 18: Byte 54, Byte 55, Byte 56
Zeile 19: Byte 57, Byte 58, Byte 59
Zeile 20: Byte 60, Byte 61, Byte 62
```

Da jedes Bit eines Bytes der Sprite-Daten einem Dot des Sprites zugeordnet ist, ergeben sich fuer die Bedeutung der Bits folgende Moeglichkeiten:

Bitwert	Bedeutung
0	transparent, keine Aenderung des Bildschirmbildes
1	Darstellung des Punktes in der Farbe des MOBs

Im Gegensatz zu den Bedeutungen der Bits bei den anderen Bildschirmmodi ergibt sich aus dem Bitwert 0 hier keine Darstellung des Punktes in der Hintergrundfarbe. Vielmehr wird dieser Spritepunkt ueberhaupt nicht beachtet, so dass die Daten, die normalerweise an dieser Stelle auf dem Bildschirm stehen, weiterhin sichtbar bleiben.

Die Farbe des MOBs, also die Farbe der Punkte, deren Datenbits gesetzt sind, wird in den Registern 39 bis 46 des VIC-II-Chips festgelegt. Diese Register liegen an den Adressen 53287 bis 53294, deren untere vier Bits (LSN) den Farbcode enthalten. In diesen Registern koennen die oberen 4 Bits unbeachtet bleiben (es muss also der Farbwert nicht mittels 'AND' und 'OR' festgesetzt werden), da diese keinerlei Funk-

tion haben. Somit kann ein Sprite in jeder der sechzehn Farben dargestellt werden.

Sprites belegen einen "geraden" Block von 64 Bytes, wovon nur die ersten 63 Bytes benutzt sind. Es stehen also innerhalb eines 16-KB-Bereichs insgesamt 256 Moeglichkeiten der Ablage von Sprite-Daten zur Verfuegung. Soll zum Beispiel ein Sprite ab der Adresse 832 abgelegt werden, so ist dies der 64-Byte-Block mit der Nummer 13 (von Nummer 0 ausgehend). Die Ablage dieser Nummern fuer die acht MOBs erfolgt, wie schon bei der Erklaerung der Videomatrix angedeutet, am Ende der 24 restlichen Bytes des 1024-Byte-Bereichs des Bildschirmspeichers. Dies ist im Normalfall der Bereich von 2040 bis 2047, wobei 2040 die Startadresse von Sprite Nummer 0 angibt. Sollten also die Daten fuer Sprite Nummer 5 ab der Adresse 832 abgelegt werden, so muesste dies dem VIC-II-Chip durch ...

```
POKE 2045, 13
```

... mitgeteilt werden. Die Angabe der Position der Sprite-Daten ist allerdings nicht genug. Jedes Sprite verfuegt ausserdem noch ueber einen Schalter, der angibt, ob dieses Sprite momentan auch auf dem Bildschirm dargestellt werden soll. Diese acht Schalter befinden sich im Register 21 (Adresse 53269) des Video-Chips. Jedes Bit (von Bit 0 bis Bit 7) ist dem entsprechenden Sprite (Spritenummer identisch mit Bitnummer) zugeordnet. Ist ein solches Bit gesetzt, so wird das Sprite entsprechend den Daten dargestellt. Ist es geloescht, so existiert dieses Sprite nicht. Es wird in keiner Form auf die Daten zugegriffen oder aehnliches.

Ausserdem laesst sich jedes Sprite sowohl in X-Richtung als auch in Y-Richtung in seiner Ausdehnung verdoppeln. Somit kann ein MOB maximal eine Ausdehnung von 48 mal 42 Punkten haben. Eine Verbesserung der Aufloesung wird damit aber nicht erreicht. Jeder Bildschirmpunkt wird lediglich in zwei Zeilen und/oder Spalten dargestellt, so dass der MOB in doppelter/vierfacher Groesse erscheint.

Diese Erweiterung in X- oder Y-Richtung wird durch die Register 29 (Adresse 53277, X-Erweiterung) und Register 23 (Adresse 53271, Y-Erweiterung) festgelegt. Auch hier sind die Bits von Bit 0 bis Bit 7 den Sprites von 0 bis 7 zugeordnet. Ist ein Bit in einem dieser Register gesetzt, so wird das entsprechende Sprite in die ausgewaehlte Richtung verbreitert.

Zusaetzlich existiert noch das Prioritaetsregister, auf das aber erst spaeter im Zusammenhang mit Kollisionen eingangen werden soll. Es befindet sich an der Adresse 53275 und wird im folgenden erst einmal unbeachtet belassen.

Nun zur Festlegung der Position des Sprites auf dem Bildschirm: Jedes Sprite kann an jeder Stelle des Bildschirms stehen. In horizontaler Richtung ergeben sich, resultierend aus der Bildschirmbreite von 320 Punkten und der Spritebreite von 24 Punkten, insgesamt 297 verschiedene Positionen, in denen das Sprite vollstaendig dargestellt ist. Dies sind die X-Werte im Bereich von 24 bis 320. Dabei beziehen sich diese Werte auf die linke obere (eventuell, je nach Sprite-Daten, auch nicht vorhandene) Ecke des Spriteblocks. Ist ein X-Wert von 24 gewaehlt, so befindet sich die linke Kante des Spriteblocks am Rand des linken Bildschirmrahmens. Die 180 vollstaendig sichtbaren Y-Werte liegen von 50 bis 229. Werden Werte ausserhalb dieses Bereiches gewaehlt, so

ist das Sprite nur teilweise sichtbar. So ist es zum Beispiel moeglich, ein Sprite langsam auf dem Bildschirm erscheinen zu lassen.

Die Festlegung der Sprite-Koordinaten geht folgendermassen vor sich:

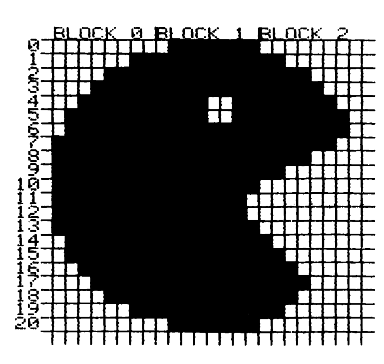
Register 1 (Adresse 53249) enthaelt die Y-Koordinate des ersten Sprites (Nummer 0). Wie bereits erwaehnt, bewirken manche Werte nur eine teilweise (oder gar keine) Darstellung des Sprites au dem Bildschirm.

Register 0 (Adresse 53248) sowie Bit 0 von Register 16 (Adresse 53264) bilden zusammen das Register fuer die X-Koordinate des Sprites Nummer 0 mit einer Breite von 9 Bits. Dies ist notwendig, da der Bildschirm eine Breite von mehr als 256 Dots hat. Bei X-Werten im Bereich von 0 bis 255 ist demnach Bit 0 von Register 16 geloescht. Ist es gesetzt, liegt der X-Wert im Bereich von 256 bis 511. Auch hier fuehren manche Werte wieder zu einer unvollstaendigen Darstellung des MOBs.

Zum langsamen Erscheinen von Sprites: Hat ein Sprite die normale Groesse von 24 mal 21 Punkten, so ist es moeglich, diesen Sprite von jeder Seite aus in einzelnen Rasterzeilen auf den Bildschirm zu bewegen. Ist der Sprite jedoch in X-Richtung verdoppelt worden, so ist er zu breit, um "unter" dem linken Rand versteckt zu werden. In X-Richtung erweiterte Sprites sollten daher nicht am linken Rand in Erscheinung gebracht werden, falls auf ein stueckweises Erscheinen Wert gelegt wird. Bei der Verbreiterung in Y-Richtung entstehen keinerlei Probleme. Auch bei verbreiterten MOBs geben die Koordinatenregister die Position der linken oberen Ecke an (eine Verbreiterung erfolgt nach rechts oder unten, nicht um den urspruenglichen Sprite herum), die linke obere Ecke veraendert ihre Position nicht.

Die Koordinaten des Sprites Nummer 1 werden in den Registern 2 (Y-Koordinate) sowie Register 3, das zusammen mit Bit 1 von Register 16 wieder den 9-Bit-breiten Wert fuer die X-Koordinate ergibt, festgelegt. Die Koordinatenregister der weiteren Sprites liegen (auch jeweils paarweise plus einem zusaetzlichen Bit aus Register 16) in den Registern 4 bis 15 (jeweils die acht niederwertigen Bits fuer die X-Koordinate im ersten Register, dann die Y-Koordinate).

Hierzu jetzt endlich ein Beispiel: Folgender Pac-Man ("Pac-Man" ist ein Warenzeichen der Firma ATARI) soll als ein Sprite definiert werden:



Um aus diesem Muster nun ein Programm zu machen, das diese Figur auf den Bildschirm bringt, muessen zuerst saemtliche 63 Bloecke zu je acht Punkten in Dezimalwerte umgewandelt werden. Dazu muss in der gleichen Weise vorgegangen werden, wie bei der Definition eigener Zeichen auch. Die daraus erhaltenen 63 Zahlen muessen dann in DATA-Zeilen eingebaut werden. Die Anordnung erfolgt wie zuvor angegeben: von Zeile 0 bis Zeile 20 und innerhalb der Zeilen die Bytes von links nach rechts. Man erhaelt fuer den "Pac-Man" folgende Werte fuer die DATAs:

```
100 DATA   0, 127,   0
110 DATA   3, 255, 192
120 DATA  15, 255, 240
130 DATA  31, 255, 248
140 DATA  63, 243, 252
150 DATA 127, 243, 254
160 DATA 127, 255, 254
170 DATA 255, 255, 252
180 DATA 255, 255, 240
190 DATA 255, 255, 192
200 DATA 255, 255,   0
210 DATA 255, 254,   0
220 DATA 255, 254,   0
230 DATA 255, 255,   0
240 DATA 127, 255, 128
250 DATA 127, 255, 192
260 DATA  63, 255, 224
270 DATA  31, 255, 240
280 DATA  15, 255, 224
290 DATA   3, 255, 192
300 DATA   0, 127,   0
```

Die folgende Zeile dient dazu, diese Werte im Bereich von 832 bis 894 abzulegen ...

```
310 FOR I = 832 TO 894 : READ A : POKE I, A : NEXT
```

Sollen nun Sprite Nummer 0 diese Daten (im Speicherblock 13) zugeordnet werden, so erreicht man dies durch ...

```
320 POKE 2040, 13
```

Ausserdem muss dieser Sprite nun noch eine Farbe (gelb) bekommen und die Freigabe fuer die Ausgabe auf dem Bildschirm muss erfolgen:

```
330 POKE 53287, 7 : POKE 53269, 1
```

Durch diese Zeile wird nun noch die Figur ueber den Bildschirm bewegt:

```
340 FOR I = 0 TO 255 : POKE 53248, I : POKE 53249, I : NEXT
```

Bevor Sie das Programm durch 'RUN' starten, sollten Sie noch einmal "RUNSTOP" und "RESTORE" druecken, damit eventuell noch vorhandene Restparameter aus anderen Programmen aus dem Video-Chip entfernt werden (Initialisierung des VIC-II-Chips). Die Figur wird nun diagonal ueber den Bildschirm bewegt.

Diese Figur kann nun natuerlich auch noch in X-Richtung und Y-Richtung vergroessert werden. Bauen Sie doch noch einmal zusaetzlich eine der folgenden Zeilen ein:

```
335 POKE 53271, 1 : POKE 53277, 0
335 POKE 53271, 0 : POKE 53277, 1
335 POKE 53271, 1 : POKE 53277, 1
```

Soll Ihr "Pac-Man" nun auch noch Fressbewegungen machen, so muessen Sie saemtliche verschiedenen Figuren erst einmal in verschiedenen 64-Byte-Bloecken ablegen. Um nun die verschiedenen Figuren auch erscheinen zu lassen, wird einfach der Zeiger auf den Datenblock (bei diesem Beispiel in Adresse 2040) der naechsten Figur gesetzt. So wird eine bewegte Spielfigur gezeigt, ohne dass die Daten ausgetauscht werden

muessen. Ein Datenblock kann jederzeit verschiedenen Sprites gleichzeitig zugeordnet sein. Besteht ein Spiel also aus mehreren gleichen Figuren, so muessen die Daten dazu nur einmal vorhanden sein.

Multicolor Sprites

Auch Sprites lassen sich im Multicolor Modus darstellen. Dies ermoeglicht es dem Programmierer, einem Sprite drei Farben (plus transparent) zuzuordnen. Jedoch werden auch hier jeweils zwei Bits der Daten verwendet, um die Farbe zu spezifizieren, so dass sich die horizontale Aufloesung auf 12 doppelte Punkte halbiert. Da aber (wie in allen Multicolor Modi) immer zwei Punkte auf einmal angesprochen werden, veraendert sich die Spritegroesse nicht. Sprites lassen sich unabhaengig voneinander in den Multicolor Modus schalten. Die geschieht durch das Setzen des entsprechenden Bits in Register 28 (Adresse 53276). Bit 0 ist hier Sprite Nummer 0 zugeordnet, Bit 7 ist fuer Sprite Nummer 7 zustaendig. Auch Multicolor Sprites lassen sich vergroessern; dies erfolgt in gleicher Weise, wie bei normalen Sprites auch.

Hier die Bedeutungen der Bitpaare:

Bitpaar	Farbquelle
00	transparent
01	MOB Multicolor #0 (Register 37)
10	MOB Farbe (Register 39 bis 46)
11	MOB Multicolor #1 (Register 38)

Wie erkennbar, existiert fuer Multicolor MOBs also eine eigene Farbe fuer jeden MOB sowie zwei globale Farben, die fuer alle MOBs verwendet werden. Diese beiden globalen Farben befinden sich in den Adressen 53285 und 53286. Auch hier koennen die Farbwerte (von 0 bis 15 fuer alle sechzehn Farben) ohne Beruecksichtigung der hoeherwertigen Bits (wie das bei allen Farbregistern der Fall ist) direkt in das Register eingeschrieben werden, da die hoeherwertigen Bits keinerlei Bedeutung haben.

Prioritaeten

Sprites haben untereinander und zu dem uebrigen Bildschirm bestimmte Prioritaeten. Ueberlappen sich zum Beispiel nichttransparente Daten zweier Sprites, so werden die Daten des Sprites mit der niedrigeren Nummer dargestellt. So verdeckt bespielsweise Sprite Nummer 0 immer die Dots der uebrigen Sprites.

Anders ist dies im Zusammenhang mit dem uebrigen Bildschirm. Hier ist es fuer den Benutzer frei waehlbar, ob die Sprites vor oder hinter den Vordergrunddaten dargestellt werden (Hintergrunddaten rangieren immer an unterster Stelle der Prioritaet und werden daher immer von nichttransparenten Daten verdeckt). Register 27 (Adresse 53275) enthaelt nun fuer jeden MOB ein Prioritaetsbit. Ist dieses Bit gesetzt, so werden MOB-Daten durch Vordergrunddaten verdeckt. Bei geloeschtem Bit haben die MOB-Daten eine hoehere Prioritaet und werden daher "ueber" den Vordergrunddaten dargestellt.

Im Zusammenhang mit den Prioritaeten erklaert sich nun auch die Spalte 'Funktion' bei den Uebersichten ueber die Bedeutungen der Bitpaare in den Multicolor Modi. Waehrend bei den Standard Modi geloeschte Datenbits Hintergrund und gesetzte Datenbits Vordergrund repraesentieren, so kann in den Multi-

color Modi mit zwei Hintergrundfarben gearbeitet werden, von denen dann auch keine (weder '00', noch '01') andere Daten verdeckt.

Dieses System hat allerdings einen kleinen Nachteil: Da bei Ueberlagerung zweier MOBs zuerst die Prioritaet der MOBs untereinander ausgewertet wird und dann erst die Prioritaet zum Bildschirm, mag es in einigen Faellen zu einer "falschen" Darstellung auf dem Bildschirm kommen, indem durch ein MOB verdeckte Vordergrunddaten durch ein zweites Sprite (mit niederer Nummer und daher hoeherer Prioritaet zum anderen Sprite, aber niederer Prioritaet zum Bildschirmvordergrund) wieder "sichtbar" gemacht werden. Dies duerfte aber in den meisten Faellen vernachlaessigt werden.

Kollisionen

In Spielen ist es wichtig festzustellen, ob sich Sprites beruehren oder ob ein Sprite soeben Vordergrunddaten ueberlappt. Alle diese Faelle werden naemlich durch den VIC-II-Chip bereits geprueft. Es muessen die fertigen Daten nur noch ausgewertet werden.

Kollisionen zwischen Sprites

Treffen nichttransparente Daten zweier (oder mehr) Sprites zusammen, so wird in Register 30 (Adresse 53278) das entsprechende Bit eines jeden Sprites gesetzt, das in dieses Zusammentreffen verwickelt ist. Gleichzeitig wird Bit 2 des Interrupt Latch Registers gesetzt, so dass (falls der Interrupt freigegeben ist) der Prozessor in die Interruptroutine verzweigen kann, wo dann die Kollision behandelt werden kann.

Kollision zwischen Vordergrunddaten und Sprites

Auch hier wird beim Zusammentreffen eines Sprites mit Vordergrunddaten (im Multicolor Modus gilt das Bitpaar '01', wie im Zusammenhang mit den Prioritaeten, als Hintergrund) fuer dieses Sprite in Register 31 (Adresse 53279) das zum Sprite gehoerige Bit gesetzt. Hierbei wird dann Bit 1 des Interrupt Latch Registers gesetzt. Eine Behandlung von Kollisionen zwischen Sprites und Vordergrunddaten kann daher auch hier durch eine Interruptroutine erfolgen.

Kollisionen (allgemein)

Ein Bit in einem der Kollisionsregister bleibt solange gesetzt, bis das Register ausgelesen wird. Das zugehoerige Bit im Interrupt Latch wird dann gesetzt, wenn mindestens ein Bit in einem der Kollisionsregister gesetzt wird. Bei folgenden Kollisionen wird dieses Latchbit NICHT gesetzt, falls nicht vorher das Kollisionsregister gelesen wurde, auch wenn das Latchbit mittlerweile geloescht wurde.

Die Interrupt-Register

Die Interrupt-Register ermoeglichen ausser der Erkennung von Kollisionen noch die Abfrage des Lightpen-Eingangs und des Rasteregisters (siehe jeweils dort).

Die Steuerung der Interrupts wird durch zwei Register des VIC-II-Chips vorgenommen: Das erste ist das Interrupt Enable Register (Register 26, Adresse 53274). In ihm wird festgelegt, ob durch einen der vier Faelle, die einen Interrupt ausloesen koennen, auch die Interrupt-Leitung des Prozessors

auf Low gezogen werden soll. Die in diesem Register verwendeten unteren vier Bits stehen fuer die vier Interruptquellen. Jede Interruptquelle kann also unabhaengig von anderen Interrupts als Ausloeser eines IRQs freigegeben werden, um so dem Prozessor mitzuteilen, dass ein zu behandelndes Ereignis aufgetreten ist.

Freigegeben wird ein Interrupt, indem das zugehoerige Bit des Interrupt Enable Registers auf den Wert 1 gesetzt wird. Das Sperren einer Interruptquelle erreicht man entsprechend durch das Loeschen des zugehoerigen Bits. Beim Auslesen wird der entsprechende Wert zurueckgegeben (die unbenutzten Bits 4 bis 7 haben den Bitwert 1)

Hier die Zuordnungen der Bits sowohl des Interrupt Latch Registers als auch des Interrupt Enable Registers:

Bitposition	Zustaendig fuer Interruptquelle ...
0	Rasterregister
1	Kollision zwischen Sprite und Vordergrunddaten
2	Kollision zwischen Sprites
3	Negative Flanke auf Lightpen-Input

Unabhaengig davon, ob eine Interruptquelle als Ausloeser freigegeben ist, wird im Interrupt Latch Register (Register 25, Adresse 53273) das zur Quelle gehoerige Bit gesetzt, wenn ein entsprechendes Ereignis auftritt (Kollision etc.). Ist ausserdem dieser Interrupt auch durch ein gesetztes Bit im Interrupt Enable Register freigegeben, so wird zusaetzlich Bit 7 im Interrupt Latch Register gesetzt (ansonsten ist Bit 7 geloescht). Dies signalisiert dem Programm, dass die Interruptleitung zum Prozessor auf Low-Pegel gelegt wurde.

Dieses Bit 7 von Register 25 laesst sich verwenden, um innerhalb der Interruptroutine zu erkennen, ob der VIC-II-Chip Ausloeser fuer den Interrupt war, da schliesslich auch die IRQ-CIA einen IRQ ausloesen kann. So kann gleich zu Beginn der Routine festgestellt werden, ob eine Verzweigung in eine Behandlungsroutine fuer den VIC-II-Chip erfolgen muss.

Ein gesetztes Latchbit im Interrupt Latch Register kann durch Schreiben einer Eins in die gleiche Bitposition wieder geloescht werden.

Das Raster Register

Dieses Register (Register 18, Adresse 53266 sowie Bit 7 von Register 17, Adresse 53265) gibt die Zeile des momentan auf den Bildschirm gebrachten Bildschirminhalts an. Dies kann dazu verwendet werden, um bei Aenderungen des Bildschirminhalts ein Flackern zu vermeiden. Man darf in diesem Fall dann nur solche RAM-Bereiche aendern, die waehrend des Aenderungsvorganges nicht durch den VIC-II-Chip ausgelesen werden.

Wird in dieses Register 18 (inklusive Bit 7 von Register 17) schreibend zugegriffen, so wird dieser Wert in ein internes Latch uebertragen. Erreicht das Raster Register diesen zwischengespeicherten Wert, so wird Bit 0 des Interrupt Latch Registers gesetzt. Ist Bit 0 des Interrupt Enable Registers gesetzt, so wird ausserdem ein IRQ ausgeloest. Das Raster Register wird beim Commodore 64 dazu benutzt, um die Fernsehnorm/Taktfrequenz des Geraetes festzustellen.

Der Lightpen-Input

Bei Auftreten einer negativen Flanke auf dem Lightpen-Input wird die momentane Bildschirmposition in ein Latch uebertragen. Das Register 19 (Adresse 53267) enthaelt dann die acht hoechstwertigen der neun fuer die X-Position zustaendigen Bits. Daher betraegt die horizontale Aufloesung nur zwei Dots. Die Y-Position zum Zeitpunkt der negativen Flanke wird in Register 20 (Adresse 53268) uebertragen (innerhalb des sichtbaren Bereichs des Bildschirms reichen fuer die vertikale Aufloesung jedoch acht Bits aus). Je Einzelbild kann aber nur einmal eine Uebertragung der Position in das Lightpen Latch erfolgen.

Bei einer negativen Flanke am Lightpen-Input wird ausserdem Bit 3 im Interrupt Flag Register gesetzt, so dass bei gesetztem Bit 3 im Interrupt Enable Register ein IRQ ausgeloest wird. Allerdings sollte in Programmen beachtet werden, dass - durch Verwendung der Ports der IRQ-CIA sowohl fuer die Tastatur als auch fuer die Kontrollports - auch durch die Tastatur eine negative Flanke auf den Lightpen-Input gegeben werden kann.

SCREEN BLANKING

Wird Bit 4 in Register 17 (Adresse 53265) geloescht, so wird der gesamte Bildschirm mit der Rahmenfarbe (Exterior Color, Register 32) gefuellt. Der Prozessor greift nur noch waehrend Phase 1 auf den Systembus zu, der Prozessor wird nicht mehr angehalten, ihm steht also der gesamte Bus zur Verfuegung.

Allerdings wird bei aktiven Sprites trotzdem auf deren Daten zugegriffen, falls Register 21 (MOB Enable) nicht auf den Wert null gesetzt wurde.

Da waehrend der Recorderoperationen die CPU keinesfalls angehalten werden darf (es handelt sich um sehr zeitabhaengige Routinen), wird beim Commodore 64 waehrenddessen der Bildschirm auf die genannte Weise "abgeschaltet", um ein einwandfreies Arbeiten des Computers zu gewaehrleisten. Sollten waehrend dieses Zeitraums jedoch MOBs aktiviert sein, so duerfte es vermutlich zu Lesefehlern kommen (falls ueberhaupt Daten erkannt wurden).

Scrolling in einzelnen Punktzeilen

Unter "Scrolling" versteht man das Verschieben des Bildschirminhalts in eine bestimmte Richtung. Waehrend der Commodore 64 den Bildschirminhalt im Normalfall nur nach oben verschieben kann (wenn der Cursor ueber den unteren Bildschirmrand hinaus bewegt wird oder ueber den unteren Bildschirmrand hinaus gedruckt wird), so kann auch die Notwendigkeit bestehen, den Bildschirminhalt in die anderen Richtungen zu verschieben. Diese Eigenschaft hat zum Beispiel ein sogenannter "SCROLLER", der zum Beispiel in EXBASIC LEVEL II fuer den Commodore 64 enthalten ist. Beim Programmieren von BASIC ermoeglicht er es, BASIC-Programme ueber den Bildschirm zu rollen, als waeren sie in einem Band vorhanden. Das englische Wort "scroll" heisst schliesslich auch "Schriftrolle". Wird mit dem Cursor ueber den Bildschirmrand hinausgefahren, so wird die naechste oder vorherige (je nachdem, ob der untere oder obere Bildschirmrand "ueberschritten" wurde) BASIC-Zeile automatisch gelistet, was die Korrektur von Programmen erheblich vereinfacht. Diese Art des Scrollens ist allerdings auch nur waehrend der Programm-

entwicklung verwendbar. Bei der Ausfuehrung eines Programms muessen andere Moeglichkeiten vorhanden sein.

Beim Scrolling, das der VIC-II-Chip des Commodore 64 unterstuetzt, handelt es sich jedoch um ein Scrolling, das den Bildschirminhalt in einzelnen Punktzeilen, aus dem die Computergraphiken aufgebaut sind, verschiebt, wohingegen das normale BASIC-Scrolling immer um ganze Zeichen - bestehend aus acht Punktzeilen - verschiebt (anders ist dies in BASIC ja auch nicht sinnvoll).

Diese Moeglichkeit des Video-Chips kann dazu benutzt werden, Daten und Informationen an den Benutzer langsam auf dem Bildschirm erscheinen zu lassen. Waehrend der VIC-II-Chip dazu jedoch schon einen erheblichen Beitrag leistet, indem er das Scrollen innerhalb eines Bereichs von acht Rasterzeilen erledigt, so muss das eigentliche Scrollen um acht Rasterzeilen (also um ein ganzes Zeichen) durch ein getrenntes Maschinenprogramm selbst erfolgen.

Hier erst einmal das Prinzip des sogenannten "Smooth Scrolling" in der Theorie:

Da die Daten, die auf den Bildschirm gescrollt werden sollen, langsam erscheinen und zeitweilig nur teilweise sichtbar sind (genauso wie die Zeichen, die langsam verschwinden), wird eine Pufferzone benoetigt, die die neuen Daten aufnimmt und dann nach und nach auf den Bildschirm gebracht wird. Diese Pufferzone in Form einer unsichtbaren Zeile oder Spalte erhaelt man, indem man den Bildschirm einfach verkleinert. Dieses ist bereits durch den VIC-II-Chip vorgesehen. Soll in horizontaler Richtung verschoben werden, so kann der Bildschirm einfach auf eine Breite von 38 Zeichen verkleinert werden. Ebenso ist es moeglich (bei vertikaler Verschiebung des Bildschirminhalts), eine Verkleinerung auf 24 Zeilen zu erreichen. Es existieren nun "geschuetzte" Zeilen, die nicht gesehen werden koennen (aber fuer den Computer immer noch vorhanden sind), da die Umrahmung nun eine groessere Flaeche als zuvor einnimmt.

Die noch sichtbare Flaeche kann nun (je nachdem was verkleinert wurde) um jeweils acht Punktzeilen (wird durch drei Bits in einem Register des Video-Chips festgelegt) verschoben werden. Bei einem Extremwert (null oder sieben, je nach Richtung) ist gerade eine Zeile (oder gar keine) des neuen Zeichens zu erkennen, beim anderen Extremwert (sieben oder null, je nach Richtung) das gesamte Zeichen (oder sieben Zeilen davon).

Nach der Verkleinerung des Bildschirms wird zuerst das Register fuer die einzelnen Zeilen auf den Maximal- oder Minimalwert gebracht (richtungsabhaengig, siehe Uebersicht). Dann werden die neuen Daten "unter" dem Rand (also dorthin, wo nur eine oder gar keine Punktzeile des Zeichens zu erkennen ist) plaziert. Darauf erfolgt die Verminderung/Erhoehung des Registers, das fuer die Einzelzeilenverschiebung zustaendig ist. Ist der Minimal- beziehungsweise Maximalwert erreicht, so wird das Maschinenprogramm aufgerufen. Es verschiebt nun den Bildschirminhalt wirklich (also nicht nur scheinbar gemaess der Darstellung) und setzt sofort darauf das Einzelzeilenregister wieder auf den Anfangswert zurueck, so dass wieder neue Daten plaziert werden koennen.

Die Verkleinerung auf eine Breite von 38 Zeichen je Zeile (nur scheinbar!!!) erfolgt durch Loeschen von Bit 3 in Register 22 (Adresse 53270) des VIC-II-Chips. Bit 3 von Register

17 (Adresse 53265) dient entsprechend zur vertikalen Schrumpfung des Bildschirms auf 24 Zeilen.

Die Einzelzeilenregister zu je drei Bits (fuer acht verschiedene Werte) befinden sich jeweils in den Bits 0 bis 2 von Register 22 (Adresse 56270) fuer die Verschiebung in X-Richtung sowie Register 17 (Adresse 53265) fuer eine Y-Verschiebung.

Hier eine Uebersicht, welche Register in welcher Weise beeinflusst werden muessen, um eine Verschiebung in eine bestimmte Richtung zu erreichen:

	oben	unten	links	rechts
Anzahl Spalten	40	40	38	38
Anzahl Zeilen	24	24	25	25
Scrollregister	Y	Y	X	X
Anfangswert	7	0	7	0
Veraenderung	-1	+1	-1	+1
Pufferrand	unten	oben	rechts	links

Befindet sich der Wert null (als Anfangswert) in einem Scrollregister, so ragt unter dem Pufferrand (fuer neue Daten) genau eine Punktzeile hervor. Wurde der Wert sieben in eins der Scrollregister (ebenfalls als Anfangswert) geschrieben, so ist das ganze Zeichen (unter dem Pufferrand) unsichtbar. Entsprechendes gilt fuer die Endwerte und Pufferraender, in denen die Daten verschwinden.

Hierzu ein Beispiel: Es soll ein auf dem Bildschirm stehender Text von unten nach oben rotiert werden, ohne dass neue Daten zugefuehrt werden.

Es wird eine Maschinenroutine benoetigt, die den Bildschirminhalt um eine Zeichenzeile nach oben verschiebt und die obere Zeile dann wieder nach unten kopiert (dies entspricht einem Rotieren des Bildschirminhalts).

Allerdings wird man bemerken, dass es damit allein nicht getan ist; wird naemlich das Einzelzeilenregister geaendert (oder die Maschinenroutine aufgerufen), so wird man bemerken, dass es zu einem Flackern auf dem Bildschirm kommt. Dies liegt daran, dass es waehrend der Erstellung des Bildes auf dem Bildschirm zu einer Aenderung der Darstellungsweise beziehungsweise der Daten kommt. Ein Teil des Bildes wird nach den alten Daten erstellt, ein anderer Teil nach neuen Daten. Dies gilt es zu verhindern. Waehrend das aber bei der Einzelzeilenverschiebung noch recht einfach ist, so bereitet es bei der Zeichenverschiebung schon Schwierigkeiten, da auch Maschinenprogramme, wenn sie auch noch so schnell sind, im Verhaeltnis zu der Zeit bei der Bilderstellung nicht schnell genug sind.

Allerdings wurde dieses Flackern durch Abfrage des Rasterzeilenregisters umgangen. Die Aenderung der Bildschirmdaten erfolgt so, dass es nicht zu einer Ueberlappung (und damit zu einem Flackern) kommen kann. Dies war aber nur durch Anwendung einer nicht sehr eleganten Methode moeglich. Aber

schliesslich kam es hierbei auf Geschwindigkeit an. Hier das fertige Programm:

```
100 FOR I = 828 TO 927 : READ A : POKE I, A : NEXT
110 DATA 120, 169, 251, 205, 18, 208, 208, 251, 206, 17, 208
120 DATA 88, 96, 120, 173, 17, 208, 16, 251, 72, 173, 17
130 DATA 208, 48, 251, 169, 107, 205, 18, 208, 208, 251, 160
140 DATA 40, 185, 255, 3, 153, 255, , 136, 208, 247, 185, 40
150 DATA 4, 153, , 4, 200, 208, 247, 185, 40, 5, 153, , 5
160 DATA 200, 208, 247, 185, 40, 6, 153, , 6, 200, 208, 247
170 DATA 160, 64, 185, 232, 6, 153, 192, 6, 200, 208, 247
180 DATA 160, 39, 185, , 1, 153, 192, 7, 136, 16, 247, 104
190 DATA 9, 7, 141, 17, 208, 88, 96
200 POKE 53265, PEEK (56265) AND 240 OR 7 : C = PEEK (646)
210 FOR I = 55296 TO 56295 : POKE I, C : NEXT : N = 63
300 FOR I = 0 TO 6 : FOR J = 0 TO N : NEXT : SYS 828 : NEXT
310 FOR J = 0 TO N : NEXT : SYS 841 : GOTO 300
```

In Zeile 100 wird das Maschinenprogramm in den Bereich des Cassettenpuffers eingelesen, da dieser normalerweise unbenutzt ist (das Programm enthaelt jedoch keinerlei Absolutadressen und kann daher auch an jeder anderen Stelle im Arbeitsspeicher stehen). Die Daten fuer das Maschinenprogramm stehen in den Zeilen 110 bis 190. Zwei aufeinanderfolgende Kommas entsprechen uebrigens dem Wert null. Zeile 200 verkleinert den Bildschirm auf 24 Zeilen und stellt die momentane Cursorfarbe fest. Mit dieser Farbe wird nun das gesamte Farb-RAM gefuellt. Mancher ahnt, was es damit auf sich hat: Die Maschinenroutine verschiebt lediglich den Inhalt der Videomatrix im Bereich von 1024 bis 2023. Das Farb-RAM und auch die Tabelle der Doppelzeilenkennzeichnungen (!) bleibt unbeachtet, da dadurch die Routine zu langsam wuerde. Daher sollte (wenn auf dem Bildschirm Programmzeilen verschoben wurden) nach Abbruch auf jeden Fall der Bildschirm geloescht werden, da es, wenn solche Programmzeilen editiert werden, zu recht merkwuerdigen Effekten kommen kann.

Die Variable N ist der Wert fuer die Geschwindigkeit (Verzoegerungsschleife). Dann wird der Bildschirm in sieben Schritten jeweils um eine Punktzeile verschoben, wozu die Routine ab der Adresse 828 dient. Sie wartet mit dem Vermindern des Einzelzeilenregisters ab, bis es zu keinem Flackern kommen kann. Die Routine ab Adresse 841 verschiebt den Bildschirminhalt um eine Zeile nach oben und setzt das Einzelzeilenregister wieder auf den Anfangswert zurueck. Auch hier wird durch eine entsprechende Abfrage ein Flackern verhindert.

Soll dieses Programm so abgeaendert werden, dass jeweils neue Daten nachgefuellt werden, so muss nach Aufruf der Verschieberoutine (SYS 841) die letzte Zeile mit den neuen Daten gefuellt werden (achten Sie darauf, dass der Cursor nicht ueber die letzte Zeile hinausfaehrt). Sind die neuen Daten "unter" dem Bildschirmrand plaziert, so kann wieder mit Zeile 300 fortgefahren werden.

Die folgenden Programme entsprechen dem obigen Programm, nur wird jeweils in eine andere Richtung verschoben. Auch jedes dieser Programme besteht aus zwei Teilen: ab Adresse 828 zum Aendern des Einzelzeilenregisters sowie ab Adresse 841 zum Verschieben um ein ganzes Zeichen. Lediglich beim Verschieben des Bildschirminhalts nach UNTEN kam es zu zeitlichen Problemen. Da hier der Bildschirminhalt von unten nach oben geaendert werden muss (im Gegensatz zu den uebrigen Programmen, die den Bildschirm von oben nach unten aendern), ist es hier nicht mehr moeglich, die Bildaenderung so zu steuern,

dass diese nicht vom Zeilenstrahl "ueberholt" wird. So existieren zwei Einsprungstellen: die erste an der Stelle 850. Hierbei kommt es zu einem Flackern, das jedoch durch geeignete Abfragen auf ungefaehr die vier unteren Bildschirmzeilen begrenzt werden kann. Die zweite Einsprungstelle befindet sich an der Adresse 841. Hier wird nun der Bildschirm "abgeschaltet" (siehe SCREEN BLANKING). Es sollte jedoch bei Verwendung dieser Einsprungstelle die Farbe fuer den Rahmen identisch mit der des Hintergrunds sein, da dann das Abschalten am wenigsten stoert.

Scrolling nach LINKS:

```
100 FOR I = 828 TO 927 : READ A : POKE I, A : NEXT
110 DATA 120, 169, 251, 205, 18, 208, 208, 251, 206, 22, 208
120 DATA 88, 96, 120, 173, 17, 208, 16, 251, 173, 17, 208
130 DATA 48, 251, 169, 60, 205, 18, 208, 208, 251, 162, 39
140 DATA 134, 251, 232, 134, 253, 162, 3, 134, 252, 134, 254
150 DATA 24, 162, 25, 160, 217, 177, 251, 72, 177, 253, 145
160 DATA 251, 200, 208, 249, 136, 104, 145, 253, 165, 251
170 DATA 105, 40, 133, 251, 165, 252, 105, , 133, 252, 165
180 DATA 253, 105, 40, 133, 253, 165, 254, 105, , 133, 254
190 DATA 202, 208, 213, 173, 22, 208, 9, 7, 141, 22, 208, 88
200 DATA 96
300 POKE 53270, PEEK (53270) AND 240 OR 7 : C = PEEK (646)
310 FOR I = 55296 TO 56295 : POKE I, C : NEXT : N = 63
320 FOR I = 0 TO 6 : FOR J = 0 TO N : NEXT : SYS 828 : NEXT
330 FOR J = 0 TO N : NEXT : SYS 841 : GOTO 320
```

Scrolling nach RECHTS:

```
100 FOR I = 828 TO 927 : READ A : POKE I, A : NEXT
110 DATA 120, 169, 251, 205, 18, 208, 208, 251, 238, 22, 208
120 DATA 88, 96, 120, 173, 17, 208, 16, 251, 173, 17, 208
130 DATA 48, 251, 169, 73, 205, 18, 208, 208, 251, 162, 255
140 DATA 134, 251, 232, 134, 253, 162, 3, 134, 252, 232, 134
150 DATA 254, 24, 162, 25, 160, 39, 177, 253, 72, 177, 251
160 DATA 145, 253, 136, 208, 249, 104, 145, 253, 165, 251
170 DATA 105, 40, 133, 251, 165, 252, 105, , 133, 252, 165
180 DATA 253, 105, 40, 133, 253, 165, 254, 105, , 133, 254
190 DATA 202, 208, 214, 173, 22, 208, 41, 248, 141, 22, 208
200 DATA 88, 96
300 POKE 53270, PEEK (53270) AND 240 : C = PEEK (646)
310 FOR I = 55296 TO 56295 : POKE I, C : NEXT : N = 63
320 FOR I = 0 TO 6 : FOR J = 0 TO N : NEXT : SYS 828 : NEXT
330 FOR J = 0 TO N : NEXT : SYS 841 : GOTO 320
```

Scrolling nach UNTEN:

```
100 FOR I = 828 TO 924 : READ A : POKE I, A : NEXT
110 DATA 120, 169, 251, 205, 18, 208, 208, 251, 238, 17, 208
120 DATA 88, 96, 120, 173, 17, 208, 41, 239, 141, 17, 208
130 DATA 120, 160, 39, 185, 192, 7, 153, , 1, 136, 16, 247
140 DATA 185, 192, 6, 153, 232, 6, 136, 208, 247, 136, 185
150 DATA 193, 5, 153, 233, 5, 136, 208, 247, 136, 185, 194
160 DATA 4, 153, 234, 4, 136, 208, 247, 160, 195, 185, 255
170 DATA 3, 153, 39, 4, 136, 208, 247, 160, 39, 185, , 1
180 DATA 153, , 4, 136, 16, 247, 173, 17, 208, 41, 248, 9
190 DATA 16, 141, 17, 208, 88, 96
300 POKE 53265, PEEK (53265) AND 240 : C = PEEK (646)
310 FOR I = 55296 TO 56295 : POKE I, C : NEXT : N = 63
320 FOR I = 0 TO 6 : FOR J = 0 TO N : NEXT : SYS 828 : NEXT
330 FOR J = 0 TO N : NEXT : SYS 841 : GOTO 320
```

Registeruebersicht fuer den VIC-II-Chip:

Die Adresse eines Registers errechnet sich aus der Summe der Registernummer und der Startadresse 53248.

Register 0: enthaelt die acht niederwertigen der insgesamt neun Bits der X-Koordinate von Sprite #0. Das hoechstwertige Bit befindet sich in Bit 0 von Register 16.

Register 1: enthaelt die Y-Koordinate von Sprite #0. Hierzu exisitiert kein weiteres Bit, da nur acht Bits zur Darstellung der Y-Koordinate benoetigt werden.

Register 2: bis Register 15: diese Register haben paarweise die gleiche Bedeutung wie die Register 0 und 1, jedoch jeweils fuer Sprite #1 bis Sprite #7.

Register 16: hier befinden sich die acht hoechstwertigen Bits der X-Koordinaten der Sprites. Dabei ist jedem Sprite das jeweilige Bit mit der Nummer des Sprites zugeordnet.

Register 17: Bits 0 bis 2 enthalten den Wert der vertikalen Verschiebung des Bildschirms in Rasterzeilen (siehe Scrolling).
Ist Bit 3 geloescht, so werden nur noch 24 Zeilen des Bildschirms dargestellt, ansonsten alle 25 Zeilen.
Ein Loeschen von Bit 4 bewirkt das Ausfuellen des gesamten Bildschirms mit der Farbe des Rahmens (Bildschirm "ausschalten").
Bit 5 schaltet (wenn gesetzt) in den Bit Map Modus um.
Bit 6 schaltet (wenn gesetzt) in den Extended Color Mode (Modus fuer erweiterte Hintergrundfarben).
Bit 7 ist das hoechstwertige Bit der neun Bits des Rasterregisters.

Register 18: Rasterregister (zusammen mit Bit 7 aus Register 17). Wird das Register gelesen, so wird die momentane Rasterzeile (die auf dem Bildschirm dargestellt wird) ausgelesen. Beim Schreiben wird der geschriebene Wert in einen internen Zwischenspeicher uebertragen, der dann zum Ausloesen eines IRQs verwendet wird.

Register 19: X-Koordinate der Position des Zeilenstrahls beim Ausloesen einer negativen Flanke auf dem Lightpen-Input (die acht hoechstwertigen der neun Bits).

Register 20: wie Register 19, aber Y-Koordinate

Register 21: Sprite Enable. Jedem Sprite ist ein Bit (der Nummer entsprechend) zugeordnet. Ist das Bit gesetzt, so ist dieses Sprite aktiviert ("eingeschaltet").

Register 22: Bits 0 bis 2 enthalten den Wert der horizontalen Verschiebung des Bildschirms in Rasterzeilen (siehe Scrolling).
Ist Bit 3 geloescht, so werden nur noch 38 Spalten des Bildschirms dargestellt, ansonsten alle 40 Spalten.
Bit 4 schaltet (wenn gesetzt) in den Multicolor Modus um.
Bit 5 ist das Reset-Bit und muss bei der Initialisierung geloescht werden, da ansonsten keinerlei Operationen von Seiten des VIC-II-Chips durchgefuehrt werden.

Register 23: Sprite Y-Expand. Wird ein Bit gesetzt, so wird das zugehoerige Sprite in doppelter Breite dargestellt.

Register 24: Bits 1 bis 3 legen die Startadresse des Zeichengenerators fest.
Im Bit Map Modus gibt Bit 3 jedoch die Startadresse der Bit Map an.
Durch Bit 4 bis 7 wird die Startadresse der Videomatrix (Bildschirmspeicher) festgelegt.

Register 25: Interrupt Flag Register:
Bit 0: Rasterregister
Bit 1: MOB-DATA Kollision
Bit 2: MOB-MOB Kollision
Bit 3: Lightpen
Ist ein Bit gesetzt, so ist die zugehoerige Bedingung erfuellt. Wird eine Eins in eine Bitposition geschrieben, so wird das Bit geloescht.
Bit 7 ist gesetzt, falls mindestens eins der Bits von 0 bis 3 zusammen mit seinem Enable Bit gesetzt ist.

Register 26: Interrupt Enable Register:
Belegung wie oben (Bit 7 unbenutzt). Zugehoeriges Bit gibt an, ob die Interruptquelle einen IRQ ausloesen kann.

Register 27: MOB-DATA Prioritaetsregister: Ist das zum Sprite gehoerige Bit gesetzt, so besitzt das Sprite eine niedrigere Prioritaet als der Vordergrund.

Register 28: MOB-Multicolor: Bei gesetztem Bit wird das zugehoerige Sprite im Multicolor-Modus dargestellt.

Register 29: Sprite X-Expand. Wird ein Bit gesetzt, so wird das zugehoerige Sprite in doppelter Hoehe dargestellt.

Register 30: MOB-MOB Collision: zum MOB gehoeriges Bit wird gesetzt, falls Beruehrung mit anderem MOB erfolgte.

Register 31: MOB-DATA Collision: zum MOB gehoeriges Bit wird gesetzt, falls Beruehrung mit Vordergrunddaten erfolgte.

Register 32: Exterior Color (Rahmenfarbe)

Register 33: Background Color #0 (Standard Hintergrundfarbe)

Register 34: Background Color #1 bis Background Color #3,
bis weitere Hintergrundfarben fuer andere Darstel-
Register 36: lungsmodi.

Register 37: enthalten die Farben zur Darstellung von Multi-
Register 38: color Sprites.

Register 39: Jedes Register enthaelt eine Farbe zur Darstel-
bis je eines Sprites (von Nummer 0, Register 39 bis
Register 46: Nummer 7, Register 46).

Bei Farbregistern (Register 32 bis 46) ist jeweils nur das untere Nybble (Bit 0 bis Bit 3) benutzt.

Nichtaufgefuehrte Bits sind unbenutzt. Unbenutzte Bits enthalten den Wert eins.

Die Farben des VIC-II-Chips

Der Video-Chip verfuegt ueber die Moeglichkeit der Darstellung von sechzehn Farben. Diese Farben werden in Form von Farbcodes sowohl in den Farbregistern als auch im Color-Nybble-RAM verwendet. Hier eine Uebersicht ueber alle Farben und wie sie ueber die Tastatur erreicht werden koennen (PRINT):

Farbcode	Farbe	Tastatur
0	schwarz	Control - 1
1	weiss	Control - 2
2	rot	Control - 3
3	cyan (tuerkis)	Control - 4
4	violett	Control - 5
5	gruen	Control - 6
6	blau	Control - 7
7	gelb	Control - 8
8	orange	Commodore - 1
9	braun	Commodore - 2
10	rosa	Commodore - 3
11	hellgrau	Commodore - 4
12	mittelgrau	Commodore - 5
13	hellgruen	Commodore - 6
14	hellblau	Commodore - 7
15	dunkelgrau	Commodore - 8

Ist der Multicolor Modus eingeschaltet (bei Zeichendarstellung), so stehen nur die ersten acht Farben fuer Zeichen zur Verfuegung.

Die Pinbelegung des VIC-II-Chips (MOS 6567)

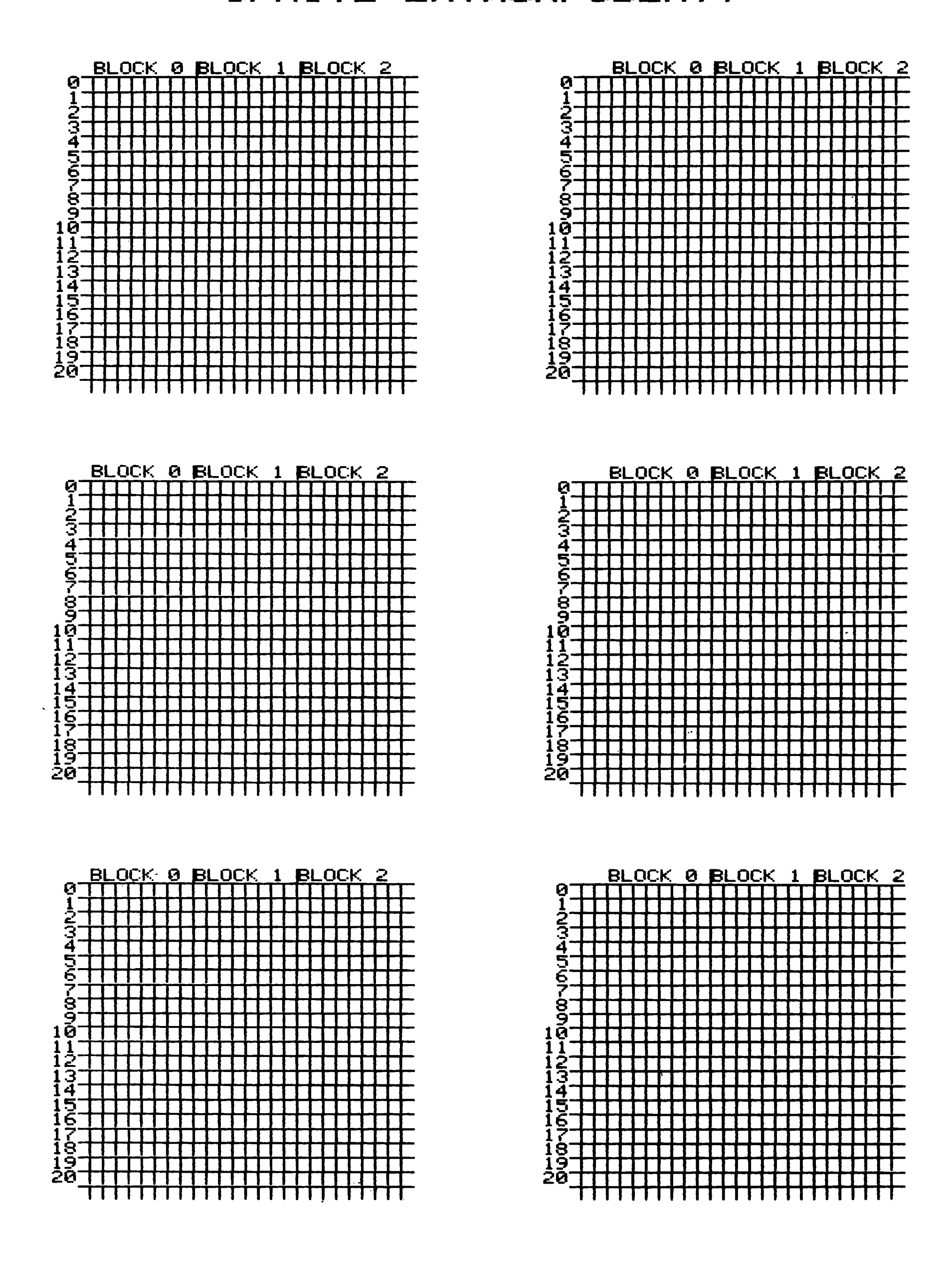
```
Pin  1 -  7: Datenbus des Prozessors (Bit 6 bis Bit 0)
Pin  8     : IRQ/ Ausgang zum Prozessor
Pin  9     : Lightpen/ Eingang
Pin 10     : CS/ Chip Select, Zugriff auf Register des Chips
Pin 11     : R/W Schreib/Leseleitung des Prozessors, 0=Write
Pin 12     : BA (Bus Available) Freigabe des Prozessorbus'
Pin 13     : VDD, Stromversorgung (+12 Volt)
Pin 14     : COLOR, Ausgang fuer Farbsignal
Pin 15     : SYNC, Ausgang fuer Synchronisationssignale
Pin 16     : AEC/, Kontrolle des Systembus' durch VIC
Pin 17     : PHO, Ausgang des Prozessortakts
Pin 18     : RAS/, Steuersignal fuer dynamische RAMs
Pin 19     : CAS/, Steuersignal fuer dynamische RAMs
Pin 20     : GND, Masseleitung
Pin 21     : PHCL, Eingang der Colorfrequenz
Pin 22     : PHIN, Eingang der Dotfrequenz
Pin 23     : All, Adressbus des Prozessors
Pin 24 - 29: A0 bis A5, A8 bis A13, gemultiplexter Adressbus
Pin 30 - 34: A6 bis 10, Adressbus des Prozessors
Pin 35 - 38: Datenbus fuer Zugriff auf Color Nybble RAM
Pin 39     : Datenbus des Prozessors (Bit 7)
Pin 40     : VCC, Stromversorgung (+5 Volt)
```

PIN CONFIGURATION

MOS 6567

Signal	Pin	Pin	Signal
DB6	1	40	VCC
DB5	2	39	DB7
DB4	3	38	DB8
DB3	4	37	DB9
DB2	5	36	DB10
DB1	6	35	DB11
DB0	7	34	A10
IRQ	8	33	A9
LP	9	32	A8
CS	10	31	A7
R/W	11	30	A6
BA	12	29	A5/A13
VDD	13	28	A4/A12
COLOR	14	27	A3/A11
SYNC	15	26	A2/A10
AEC	16	25	A1/A9
PH0	17	24	A0/A8
RAS	18	23	A11
CAS	19	22	PHIN
GND	20	21	PHCL

SPRITE-ENTWURFSBLATT



BEDIENUNG DES SPRITE-GENERATORS

Das Programm "SPRITE GENERATOR" wandelt nach dem Start durch "RUN" ein in DATA-Zeilen befindliches Muster der Groesse von 21 Zeilen mal 24 Spalten in die entsprechenden Dezimalwerte fuer die Definition eines Sprites um. Die Dezimalwerte werden automatisch in Programmzeilen mit DATAs umgewandelt, deren erste Zeilennummer durch die Variable "L" und der Abstand zwischen den Zeilen durch "S" (beides in Zeile 1) festgelegt werden. Es muss natuerlich darauf geachtet werden, dass die neuen Zeilennummern nicht das Ursprungsprogramm ueberschreiben, wenn noch weitere Sprites kreiert werden sollen.

Die Variable "F" (ebenfalls in Zeile 1) legt fest, ob das Muster als Standard- (F=1) oder Multicolor-Sprite (F=2) aufgefasst werden soll. Im Standardmodus entspricht ein Leerzeichen dem Bitwert 0 und ein Asterisk "*" einem gesetzten Bit. Im Gegensatz dazu werden im Multicolormodus immer zwei aufeinanderfolgende Zeichen behandelt. So entsprechen zwei Blanks dem Bitpaar "00", "**" entspricht "01", "++" wird in "10" und "##" in "11" umgewandelt.

SPRITE GENERATOR

```
0 REM ### SPRITE GENERATOR FOR THE COMMODORE 64 ###
1 L=100:S=10:F=1:PRINT"■■":N=.:GOSUB6:FORI=.TO20:READA$:FORJ=.TO2:B=.
2 FORK=1TO8STEPF:B=2*B*F:C$=MID$(A$,J*8+K,F):IFC$="*"ORC$="**"THENB=B+1
3 B=B-2*(C$="++")-3*(C$="##"):NEXT:P$="":IFBTHENP$=MID$(STR$(B),2)
4 IFPOS(.)+LEN(P$)>75THENPRINTCHR$(20):GOSUB6
5 PRINTP$",";:NEXT:NEXT:PRINTCHR$(20)"■";:POKE198,N:END
6 PRINTMID$(STR$(L),2)"D♠";:L=L+S:POKE631+N,13:N=N+1:RETURN
10 DATA"         *******        "
11 DATA"      ************      "
12 DATA"    ****************    "
13 DATA"   ******************   "
14 DATA"  **********  ********  "
15 DATA" ***********  ********* "
16 DATA" ********************** "
17 DATA"**********************  "
18 DATA"********************    "
19 DATA"******************      "
20 DATA"****************        "
21 DATA"***************         "
22 DATA"***************         "
23 DATA"****************        "
24 DATA" ****************       "
25 DATA" *****************      "
26 DATA"  *****************     "
27 DATA"   *****************    "
28 DATA"    ***************     "
29 DATA"      ************      "
30 DATA"         *******        "
EADY.
```

ABFRAGE UND BELEGUNG DER FUNKTIONSTASTEN

Sowohl Commodore 64 als auch VIC-20 verfuegen ueber vier Funktionstasten, die zum Beispiel die Erstellung von Programmen wesentlich vereinfachen koennen. Auch ermoeglichen sie eine bequeme und uebersichtliche Benutzerfuehrung durch Anwender- und Spielprogramme. Im folgenden die Erklaerung der praktischen Ausfuehrung dieser Moeglichkeiten:

Abfrage der Funktionstasten

Die Funktionstasten sind Tasten, deren ASCIIs im Bereich von 133 bis 140 liegen. Dies sind Codes im Bereich der nichtdruckenden Steuercodes (siehe Tabelle), was Grund dafuer ist, dass das Druecken dieser Tasten keinerlei Wirkung zeigt. Lediglich innerhalb von Anfuehrungszeichen (im sogenannten Quote-Modus) und nach dem Einfuegen von Leerzeichen durch "INSERT" (im Insert-Modus) werden Zeichen ausgegeben, abhaengig vom Darstellungsmodus entweder Balken oder Viertelkreise (im Grafikmodus) oder Buchstaben von "E" bis "L" (im Kleinschriftmodus), beide Male jedoch in Negativdarstellung. Dieses Verhalten ist aehnlich dem der Tasten zur Cursor- und Farbsteuerung.

Um nun eine gedrueckte Funktionstaste in BASIC-Programmen zu erkennen, muss entsprechend der Abfrage von anderen Tasten vorgegangen werden. Zum Beispiel so, wie im folgenden Programmausschnitt:

```
...
350 PRINT CHR$ (147) "F1 - SEGMENT A"
360 PRINT : PRINT "F3 - SEGMENT B"
370 PRINT : PRINT "BITTE 'F1' ODER 'F3' DRUECKEN!"
380 GET G$ : IF G$ = CHR$ (133) GOTO 500
390 IF G$ <> CHR$ (134) GOTO 380
400 REM SEGMENT B
...
500 REM SEGMENT A
...
```

Hier die Codes der Funktionstasten:

F1: 133	F3: 134	F5: 135	F7: 136
F2: 137	F4: 138	F6: 139	F8: 140

Die Codes von 133 bis 136 werden beim Druecken der Funktionstaste alleine erzeugt, die ASCIIs von 137 bis 140 ergeben sich beim gemeinsamen Druecken entweder mit einer SHIFT-Taste oder der COMMODORE-Taste.

Belegung von Funktionstasten

Unterschiedlich dazu ist die Vorgehensweise zum Belegen der Funktionstasten. Hierbei werden den Tasten bestimmte Texte oder Befehlsworte zugeordnet, die haeufig benoetigt werden. Bei Ausuebung dieser Funktion werden die Tasten auch als "Softkeys" bezeichnet.

Das Belegen ist NUR durch Verwendung eines geeigneten Maschinenprogramms moeglich und NICHT ueber BASIC. Hierzu wird der Sprungvektor (655/656) geaendert, so dass auf ein eigenes Maschinenprogramm verzweigt wird. Dieser Vektor wird in der Routine SCNKEY verwendet, die die Abfrage der Tastatur erledigt und normalerweise durch die Interruptroutine aufgerufen wird. SCNKEY belegt den Bereich von 60039 bis 60600 (siehe ROM-Listing).

Bei Erreichen des Sprungvektor (655/656) sind bereits die Adressen 203 und 653 auf den aktuellen Wert gesetzt worden. Erstere enthaelt den Tastaturmatrixcode von 0 bis 64, (653) gibt an, welche Kombinationstasten gedrueckt sind (SHIFT, COMMODORE und CONTROL).

Ist durch das zusaetzliche Maschinenprogramm nun festgestellt worden, dass eine Funktionstaste gedrueckt wurde, so muss nun noch der Zustand der SHIFT-Flag geprueft werden. Daraus errechnet sich dann, welcher Text im Tastaturpuffer abgelegt werden soll.

Dies wird durch das im Anschluss abgedruckte BASIC-Programm erledigt. Es legt das Maschinenprogramm zur Abfrage der Funktionstasten im Bereich ab Adresse 828 ab, kann jedoch auch in einen beliebigen anderen Bereich gelegt werden. Hierzu muss nur die Variable "S" geaendert werden. Der Beginn der Texttabelle wird durch "D" festgelegt. Sie ist in acht Eintraege zu je zehn Bytes aufgeteilt, entsprechend "Fl" bis "F8". Die Belegungen koennen durch Aendern der letzten DATA-Zeile eigenen Anwendungen angepasst werden. Der Rueckwaertspfeil gibt an, ob die RETURN-Taste gedrueckt werden soll. Die Laenge des Textes darf zehn Buchstaben nicht ueberschreiten.

Nach Ausfuehrung des Programms durch "RUN" wird die erweiterte Tastaturabfrage automatisch aktiviert. Jedoch wird Vektor (655/656) beim Druecken von "RUNSTOP" und "RESTORE" wieder auf den Normalwert gesetzt, so dass eine Neuaktivierung notwendig ist. Dies erfolgt durch einen "SYS" mit der Zahl als Argument, die beim Start des Programms ausgegeben wurde.

```
100 REM ## FUNCTION KEYS FOR THE COMMODORE 64 & THE VIC20 ##
110 S = 828 : D = 912 : FOR I = S TO S + 83 : READ A
120 POKE I, A : NEXT : POKE S + 74, S - 256 * INT (S / 256)
130 POKE S + 76, S / 256 : POKE S + 46, D / 256
140 POKE S + 45, D - 256 * INT (D / 256)
150 FOR I = 0 TO 7 : READ A$ : FOR J = 1 TO LEN (A$)
160 B = ASC (MID$ (A$, J, 1)) : IF B = 95 THEN B = 13
170 POKE D + I * 10 + J - 1, B : NEXT
180 IF I <> 11 THEN POKE D + I * 10 + J - 1, 0
190 NEXT : PRINT S + 73 : SYS S + 73
200 DATA 164, 203, 196, 197, 208, 11, 44, 250, 255, 16, 3
210 DATA 76, 220, 235, 76, 72, 235, 177, 245, 201, 137, 176
220 DATA 239, 233, 132, 144, 235, 168, 173, 141, 2, 74, 152
230 DATA 42, 133, 197, 10, 10, 101, 197, 10, 168, 162, , 185
240 DATA , , 240, 9, 157, 119, 2, 200, 232, 224, 10, 208
250 DATA 242, 134, 198, 162, 255, 44, 250, 255, 16, 3, 76
260 DATA 186, 235, 76, 38, 235, 162, , 160, , 142, 143, 2
270 DATA 140, 144, 2, 96
300 DATA RUN←,GOTO,LIST←,?FRE(0)←,POKE,PEEK(,THEN,INPUT
```

SOUNDERZEUGUNG MIT DEM COMMODORE 64

Der Commodore 64 besitzt einen hochwertigen Soundbaustein, den MOS 6581, auch SID (Sound Interface Device) genannt. Mit diesem lassen sich Geraeusche fast aller Art erzeugen, Musikstuecke spielen und Soundeffekte zum Beispiel in Spiele einbauen. Hier eine grobe Funktionsuebersicht:

Der SID verfuegt ueber drei getrennte Soundgeneratoren (die miteinander synchronisiert oder ringmoduliert werden koennen) mit einem Frequenzbereich von 0 Hz bis ca. 4 kHz. Die Huellkurve (ADSR) eines jeden Tongenerators kann getrennt gewaehlt werden, ebenso die Wellenform, von der vier verschiedene (Dreieck, Saegezahn, Rechteck und Rausch) zur Auswahl stehen. Es steht ausserdem ein Filter zur Verfuegung, mit dem die drei Tongeneratoren sowie eine externe Signalquelle verfremdet werden koennen. Der dritte Tongenerator kann ausserdem durch zwei weitere Register kontrolliert werden: Eines gibt die Huellkurve und damit die Amplitude an, das zweite Register kann zur Modulation der anderen beiden Soundgeneratoren oder zum Beispiel zur Erzeugung von Zufallszahlen dienen.

Jeder der drei Tongeneratoren kann durch folgende Parameter gesteuert werden:

Frequenz: Die Frequenz wird durch einen Wert einer Breite von 16 Bits festgelegt und wird daher durch zwei Register (je Stimme) bestimmt. Dadurch existieren 65536 verschiedene Tonhoehen. Die Frequenz errechnet sich dabei wie folgt:

F = N * C / 2 ↑ 24

Dabei ist N die 16-Bit-Zahl, C die Systemtaktfrequenz und F die effektive Frequenz (die letzteren beiden Angaben in Hertz). Um zum Beispiel einen Ton einer bestimmte Frequenz erklingen zu lassen, kann prinzipiell folgendermassen vorgegangen werden:

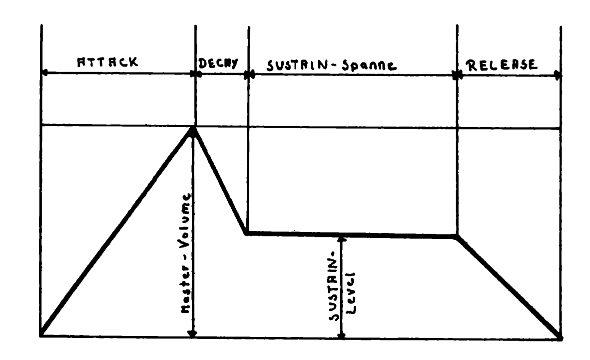
```
300 C = 14318180 / 14 : IF PEEK (678) THEN C = 17734472 / 18
310 Q = 2 ↑ 24 / C
320 N = F * Q : HI = INT (N / 256) : LO = N - 256 * HI
```

Wie ersichtlich, sollten hier, wie bei anderen Gelegenheiten auch, die verschiedenen Taktfrequenzen der verschiedenen Geraete beachtet werden, da die Taktfrequenz nicht exakt ein MHz betraegt. Die Variablen "LO" und "HI" enthalten nach Ausfuehrung nun das LSB und MSB fuer die beiden Register zur Festlegung der Frequenz. Die Frequenz laesst sich in Schrittweiten von ungefaehr 0.06 Hz festlegen.

ADSR-Funktion: Durch diese aus vier Parametern bestehende Funktion wird der Lautstaerkeverlauf gesteuert. Der erste Parameter (A, ATTACK) gibt die Zeitspanne an, in der der Ton die hoechste Lautstaerke erreicht haben soll. Ist diese Zeitspanne vorueber, so wird der DECAY-Zyklus eingeleitet. Durch diesen zweiten Parameter wird die Zeitspanne festgelegt, in der der Ton auf die durch den Parameter SUSTAIN festgelegte Lautstaerke zurueckfaellt. Diese Lautstaerke wird nun solange gehalten, bis durch Loeschen einer bestimmten Flag RELEASE beginnt. Der RELEASE-Parameter gibt, aehnlich wie bei DECAY auch, die Zeitspanne an, innerhalb der der Ton auf die Lautstaerke 0 zurueckgegangen sein soll.

Durch diese Funktion laesst sich der Klang der verschiedensten Instrumente imitieren.

Graphisch laesst sich diese ADSR-Funktion wie folgt darstellen:



Als Zeitspannen von ATTACK, DECAY und RELEASE stehen folgende Moeglichkeiten zur Auswahl:

Wert	ATTACK-Spanne	DECAY/RELEASE-Spanne
0	0.002 sec	0.006 sec
1	0.008 sec	0.024 sec
2	0.016 sec	0.048 sec
3	0.024 sec	0.072 sec
4	0.038 sec	0.114 sec
5	0.056 sec	0.168 sec
6	0.068 sec	0.204 sec
7	0.08 sec	0.24 sec
8	0.1 sec	0.3 sec
9	0.25 sec	0.75 sec
10	0.500 sec	1.5 sec
11	0.800 sec	2.4 sec
12	1 sec	3 sec
13	3 sec	9 sec
14	5 sec	15 sec
15	8 sec	24 sec

Auch wenn fuer DECAY und RELEASE die gleiche Einteilung vorhanden ist, so lassen sich diese beiden Parameter natuerlich unterschiedlich waehlen. Jedoch beziehen sich auch diese Angaben wieder auf eine Taktfrequenz von einem MHz. Sollte eine genaue Angabe der wirklichen Zeitspanne erforderlich sein, so sind die obigen Werte mit dem Faktor ...

1 MHz / Systemtaktfrequenz

... zu multiplizieren.

Als eine weitere Moeglichkeit der Beeinflussung der einzelnen Stimmen steht die Wellenform zur Verfuegung. An Wellenformen stehen vier verschiedene zur Auswahl: die Dreieck-Wellenform, die Saegezahn-Wellenform, die Pulse-Wellenform sowie die "Wellenform" Rausch, die beispielsweise fuer Explosionen, Wind und Schlagzeug genutzt werden kann.

Dreieck Saegezahn Pulse



"Rausch" liefert ein Signal in Form eines Rauschens, dessen Frequenzen innerhalb eines bestimmten Bereichs liegen.

Die Wellenform "Pulse", auch Rechteckschwingung genannt, kann ausserdem noch durch die Pulsebreite variiert werden. Dieser Wert mit einer Breite von 12 Bits im Bereich von 0 bis 4095 (die Randwerte allerdings bewirken, dass kein Ton ausgegeben wird) gibt an, wie lange das Signal den Nullpegel annehmen soll. Ist also ein Wert von 2048 gewaehlt, so erhaelt man eine exakt regelmaessige Rechteckschwingung. Durch Aendern der Pulsebreite (waehrend der Ton erklingt) koennen klangliche Veraenderungen (ein sogenanntes "Phasing") erzeugt werden. Ist eine andere Wellenform als Rechteck ausgewaehlt, so hat die Angabe einer Pulsebreite keinerlei Auswirkungen auf die Tonerzeugung.

Jede Stimme besitzt sieben Steuerregister (von den insgesamt 29 Registern des SID) zur Kontrolle. Die ersten beiden Register eines der drei Bloecke (fuer jeden Oszillator) bestimmen die Frequenz der Stimme. Dabei handelt es sich um den unter "Frequenz" beschriebenen 16-Bit-Wert, der in der Reihenfolge LSB/MSB in den beiden Registern abgelegt wird.

Die folgenden beiden Register sind fuer die Angabe der Pulsebreite verwendet. Da diese jedoch nur 12 Bits benoetigt, ist das MSN des zweiten Registers unbenutzt. Das erste enthaelt die Bits 0 bis 7, das LSN des zweiten Registers Bit 8 bis Bit 11.

Bei den Registern 4, 11 und 18 (je nach Tongenerator) handelt es sich um die Steuerregister fuer den Ablauf der Huellkurve (darauf wird im folgenden noch eingegangen). Die Bits 4 bis 7 waehlen die Wellenform aus. Ist ein bestimmtes Bit gesetzt, so wird die entsprechende Wellenform fuer den Tongenerator verwendet. Auch ist es moeglich, mehrere Bits gleichzeitig zu setzen. Allerdings resultiert die dabei entstehende Wellenform aus einer "UND"-Verknuepfung der ausgewaehlten Wellenformen. Hier eine Aufstellung, welches Bit welche Wellenform auswaehlt (sofern gesetzt):

Bit 4: Triangle (Dreieck)
Bit 5: Sawtooth (Saegezahn)
Bit 6: Pulse (Rechteck)
Bit 7: Noise (Rauschen)

Durch das naechste Register (5, 12, 19) werden die Parameter ATTACK und DECAY festgelegt. Die Daten fuer ATTACK befinden sich im MSN, DECAY belegt das LSN. Der Wert kann also durch ...

```
POKE 54272 + T * 7 + 5, A * 16 + D
```

... abgespeichert werden, wobei 54272 die Basisadresse des SID ist, T der Tongenerator (von 0 bis 2) und A und D (jeweils von 0 bis 15) die Daten fuer ATTACK und DECAY enthalten.

Das letzte tongeneratorspezifische Register dient zur Festlegung des SUSTAIN-Levels und der Zeitspanne fuer RELEASE. Die Laustaerke fuer SUSTAIN befindet sich im MSN, die RELEASE-Zeitspanne muss im LSN festgelegt werden.

Alle diese Register koennen nur BESCHRIEBEN und NICHT ausgelesen werden. Sollen also auf bereits festgelegten Parametern aufbauend andere Effekte erreicht werden, so muessen die Daten zusaetzlich im RAM abgelegt werden. Die Register befinden sich an den Adressen 54272 bis 54292.

Um nun einen Ton zu erzeugen, muss folgendermassen vorgegangen werden: Zuerst sollten die ADSR-Daten und die Frequenz (sowie, falls erforderlich, die Pulsebreite) abgespeichert werden. Ist dies erfolgt, so kann nun, gleichzeitig mit der Definition der Wellenform, der ADSR-Zyklus begonnen werden. Dies erfolgt durch Abspeichern eines Wertes in Registeradresse 54276, 54283 oder 54290 (tongeneratorabhaengig), in dem Bit 0 gesetzt ist. Bit 4 bis Bit 7 geben die Wellenform an. Ist der SUSTAIN-Level erreicht worden, so wird dieser so lange gehalten, wie das Bit 0 (GATE- oder KEY-Bit genannt) gesetzt ist. Wird es geloescht (die Bitkombonation fuer die Wellenform muss erhalten bleiben!), so wird RELEASE ausgefuehrt. Ist der Nullevel erreicht worden, kann dem Register der Wert 0 zugewiesen werden. Wird das GATE-Bit geloescht, bevor DECAY abgeschlossen ist, so wird sofort RELEASE eingeleitet. Auch wird ATTACK begonnen, falls GATE gesetzt wird, unabhaengig davon, ob sich dieser Tongenerator vielleicht noch in der RELEASE-Phase befindet. Das GATE-Bit kann so zum Beispiel in Abhaengigkeit davon, ob eine bestimmte Taste gedrueckt ist, gesetzt und geloescht werden, die Tastatur wuerde also zu einer Art Klaviatur umfunktioniert.

Um jedoch ein hoerbares Ergebnis zu erzielen, muss die Hauptlautstaerke festgesetzt werden. Diese kann in 16 Stufen von 0 (kein Ton) bis 15 (hoechste Lautstaerke) durch das LSN von Register 24 variiert werden.

Zurueck zum Kontrollregister, mit dem der ADSR-Zyklus initialisiert und die Wellenform festgelegt wird: Bit 1 dient, falls gesetzt, zur Synchronisation der Grundfrequenzen verschiedener Oszillatoren. Dazu muss der Oszillator, nach dem synchronisiert wird, auf eine von null verschiedene Frequenz gesetzt werden, andere Register dieses Oszillators haben keinen Einfluss auf die synchronisierte Stimme.

Ein Setzen von Bit 2 bewirkt, falls die Dreieck-Wellenform des entsprechenden Generators gewaehlt ist, dass der Oszillatorausgang dieser Stimme durch ein durch Ringmodulation (Summe und Differenz der beiden Grundstimmen) kombiniertes Signal dieses und eines zweiten Generators ersetzt wird. Auch hier muss der zweite Generator eine von null verschiedene Frequenz haben um einen Unterschied zur normalen Ausgabe zu erzielen.

Bit 3 sperrt den entsprechenden Oszillator und fuehrt gleichzeitig einen RESET des entsprechenden Oszillators durch. Dies kann notwendig sein, wenn bei der Auswahl der Wellenformen NOISE mit einer weiteren Wellenform kombiniert wurde, da dann der Rauschgenerator blockieren kann.

FILTER

Register 21 und 22 bilden ein Register, das die Filterfrequenz bestimmt. Bit 0 bis 2 von Register 21 (Adresse 54293), Bit 0 bis 2 des 11 Bits breiten Werts, sowie alle Bits von Register 22 (Adresse 54294), Bit 3 bis 10 des Werts, bestimmen die Filterfrequenz. Diese errechnet sich durch ...

$F = (30 + N * 5.\overline{81})$ Hz

Dabei ist N der 11-Bit-Wert und F die Filterfrequenz in Hertz. Im Gegensatz zu anderen Werten ist dieser Wert unabhaengig von der Taktfrequenz. Als Filterarten stehen nun folgende Moeglichkeiten zur Auswahl: LOWPASS (alle Frequenzkomponenten des Signals UEBER der Filterfrequenz werden mit 12 dB je Oktave abgeschwaecht), HIGHPASS (entsprechend Low-

pass, jedoch werden alle Frequenzkomponenten des Signals UNTER der Filterfrequenz mit 12 dB/Oktave abgeschwaecht) sowie BANDPASS (alle Frequenzkomponenten UNGLEICH der Filterfrequenz werden um 6 dB/Oktave vermindert).

Die verschiedenen Ausgabesignale der Filter koennen durch gleichzeitiges Setzen der entsprechenden Bits additiv verknuepft werden. So kann durch Kombination von Lowpass und Highpass eine Bandsperre programmiert werden. Der Filter wird im allgemeinen dazu verwendet, um bestimmte Frequenzen eines Signals zu eliminieren.

Ausserdem kann die Filterresonanz durch einen Wert von 0 (keine Wirkung) bis 15 (hoechste Resonanz) im MSN von Register 23 festgelegt werden. Dies bewirkt eine besondere Betonung der Frequenzkomponenten der Filterfrequenz woraus ein schrillerer Ton resultiert.

Vier verschiedene Stimmen koennen ueber den Filter laufen: jeder der Oszillatoren 1 bis 3 sowie ein externes Signal. Dieses externe Signal wird durch den Eingang "AUDIO IN" (Pin 5) ueber die Audio/Video-Buchse zum SID geleitet. Soll eine dieser Stimmen durch den Filter modifiziert werden, so muss eines der Bits des LSN von Register 23 (Bits 0 bis 2 fuer Stimme 1 bis 3, Bit 3 fuer das externe Signal) gesetzt werden. Es ist moeglich, mehrere Stimmen gleichzeitig durch den Filter zu leiten.

TONGENERATOR 3

Stimme 3 (Register 14 bis 20, Adressen 54286 bis 54292) nimmt eine gesonderte Stellung innerhalb der drei Oszillatoren ein. Hier kann der Verlauf der Huellkurve und der Zustand des Oszillators festgestellt werden.

Huellkurve: Die Huellkurve (Register 28) gibt die momentane Amplitude der ADSR-Funktion an. Ist die durch das LSN von Register 24 festgelegte Lautstaerke nach Initialisierung des ADSR-Zyklus' (Setzen des GATE-Bits) erreicht worden (nach der ATTACK-Phase), so enthaelt dieses Register den Wert 255, nach RELEASE den Wert 0. So kann zum Beispiel, falls SUSTAIN eine festgelegte Zeitspanne gehalten werden soll, nach Ablauf dieser Zeitspanne automatisch RELEASE eingeleitet werden. Dazu muss lediglich gewartet werden, bis nach dem Erreichen des Werts 255 das Huellkurvenregister den von SUSTAIN abhaengigen Inhalt hat. Dann wird die gewuenschte Zeitspanne gewartet, und nach Beendigung der Warteschleife das GATE-Bit geloescht. Auch kann so erkannt werden, wann RELEASE beendet ist, um das Kontrollregister (Register 4, 11, 18 sowie Adressen 54276, 54283, 54290) einer Stimme auf den Wert null zu setzen, um (sehr schwaches) Nachklingen zu vermeiden. Auch koennen durch dieses Register sehr interessante Effekte erzielt werden, wenn in Abhaengigkeit vom Zustand der Huellkurve die Frequenz oder Pulsebreite geaendert wird. Auch durch Aendern der Filterfrequenz sind viele wirkungsvolle Effekte moeglich.

Oszillator: Dieses Register (#27, Adresse 54299) gibt den Zustand der acht hoechstwertigen Bits von Oszillator 3 an. Der Registerinhalt aendert sich je nach gewaehlter Wellenform. Bei der Wellenform "Dreieck" nimmt der Inhalt des Registers gleichmaessig von 0 bis 255 zu. Ist der Wert 255 erreicht worden, so wird dieser Wert nun wieder bis auf null vermindert, woraufhin wieder von vorne begonnen wird. "Saegezahn" laesst den Inhalt dieses Registers auch von 0 bis 255 ansteigen, faellt nach Erreichen dieses Werts jedoch so-

fort wieder auf null zurueck. Ein staendiger Wechsel von 0 auf 255 (und zurueck) wird bei "Pulse" ausgefuehrt. Die Laenge dieser beiden Phasen ist vom Pulsebreitenregister abhaengig. Bei "Noise" koennen "zufaellige" Werte im Bereich von 0 bis 255 ausgelesen werden. Die Hauptanwendung dieses Register besteht, wie beim vorherigen Register auch, Echtzeit-Effekte auszufuehren (was natuerlich im Normalfall nur von Maschinensprache aus moeglich ist), wobei die Parameter anderer Oszillatoren durch Werte, die durch Oszillator 3 erzeugt wurden, geaendert werden.

Viele Effekte, die auf von Oszillator 3 errechneten Daten aufbauen, benoetigen diese Stimme jedoch ueberhaupt nicht. Daher ist es moeglich, Stimme 3 abzuschalten, ohne jedoch die internen Vorgaenge zu beeinflussen. Das Signal von Tongenerator 3 erscheint so also nicht am Audio-Ausgang des SID-Chips. Jedoch bleiben Synchronisation und Ringmodulation davon unbetroffen, da diese kombinierten Signale davon unabhaengig sind, ob Oszillator 3 ein- oder ausgeschaltet ist. Das Ausschalten von Stimme 3 erfolgt durch Setzen des MSB in Register 24 (Adresse 54296).

Die AD-Wandler

Der SID verfuegt ausserdem ueber zwei AD-Wandler (POTX und POTY, Adressen 54297 und 54298), die es ermoeglichen, Potentiometerzustaende zu erkennen. Die in den Registern befindlichen Werte - sie werden alle 512 Taktzyklen auf den neuesten Wert gebracht - koennen im Bereich von 0 (niedrigster Widerstand) bis 255 (hoechster Widerstand) liegen.

Beim Commodore 64 finden diese beiden Eingaenge des SID im Normalfall fuer die Paddles Verwendung, jedoch auch eine Verwendung als Temperaturfuehler oder Helligkeitsmesser waere nicht abwegig.

Registeruebersicht (die Basisadresse des SID ist 54272):

Register 0: LSB der Frequenz des Oszillators von Stimme 1

Register 1: MSB der Frequenz des Oszillators von Stimme 1

Register 2: niederwertigsten acht Bits des Pulsebreitenregisters (Stimme 1) fuer die Erzeugung von Rechtecksignalen

Register 3: das LSN dieses Registers bildet die Bits 8 bis 11 des Pulsebreitenregisters (Stimme 1), das MSN ist unbenutzt

Register 4: Kontrollregister fuer Stimme 1
Bit 0: GATE, Stimme 1
Wird dieses Bit gesetzt, so wird der durch die ADSR-Parameter gewaehlte Ablauf der Huellkurve begonnen. Innerhalb der durch ATTACK festgelegten Zeitspanne steigt die Lautstaerke bis auf den Maximalwert (durch LSN in Register 24 festgelegt), um dann die RELEASE-Phase zu beginnen, in der die Lautstaerke auf den durch SUSTAIN festgelegten Pegel zurueckfaellt. Dieser wird so lange gehalten, bis das GATE-Bit wieder rueckgesetzt wird. Im gleichen Augenblick beginnt SID RELEASE und laesst dadurch den Ton bis auf Nullpegel zurueckfallen.

Bit 1: SYNC
Eine Synchronisation von Stimme 1 mit Stimme 3 erfolgt durch Setzen dieses Bits.

Bit 2: RING MOD
Eine Ringmodulation von Stimme 1 mit Stimme 3 erfolgt durch Setzen dieses Bits.

Bit 3: TEST
Wird dieses Bit gesetzt, so wird Stimme 1 gesperrt und gleichzeitig ein Reset des Oszillators durchgefuehrt, wodurch eine eventuelle (durch Kombination von "Rausch" mit einer weiteren Wellenform) Blockade des Rauschgenerators aufgehoben wird.

Bit 4: Auswahl der Dreieckswellenform (TRIANGLE), falls Bit gesetzt

Bit 5: Auswahl der Saegezahnwellenform (SAWTOOTH), falls Bit gesetzt

Bit 6: Auswahl der Wellenform PULSE, falls Bit gesetzt

Bit 7: Auswahl der "Wellenform" NOISE, falls Bit gesetzt

Register 5: ATTACK/DECAY
Bits 4 bis 7 (ATTACK) geben die Dauer (von 2 msec bis 8 sec) an, die SID benoetigt, bis die Maximallautstaerke vom Nullpegel aus erreicht werden soll.
Bits 0 bis 3 (DECAY) definieren die Zeitspanne (von 6 msec bis 24 sec), in der die Lautstaerke vom Maximalwert (nach ATTACK) auf SUSTAIN zurueckgeht.

Register 6: SUSTAIN/RELEASE
Bits 4 bis 7 (SUSTAIN): Lautstaerkepegel, auf den bei DECAY zurueckgegangen wird. Dieser Level wird gehalten, bis GATE rueckgesetzt wird.
Bits 0 bis 3 (RELEASE): Zeitspanne, in der der Lautstaerkepegel von SUSTAIN wieder auf null zurueckgeht. Der Bereich erstreckt sich, wie bei DECAY auch, von 6 msec bis 24 sec.

Register 7 - Register 13: Diese sieben Register haben die gleiche Funktion, wie die ersten sieben Register auch, jedoch fuer Stimme 2. Ringmodulation und Synchronisation erfolgen mit Stimme 1.

Register 14 - Register 20: Diese sieben Register haben die gleiche Funktion, wie die ersten sieben Register auch, jedoch fuer Stimme 3. Ringmodulation und Synchronisation erfolgen mit Stimme 2.

Register 21: FILTER LO
Die niederwertigsten drei Bits bilden Bit 0 bis Bit 2 der Filterfrequenz (fuer alle Tongeneratoren identisch). Uebrige Bits sind ungenutzt.

Register 22: FILTER HI
Dieses Byte bildet zusammen mit Bits 0 bis 2 von Register 21 die Filterfrequenz. Diese gibt die Eck- beziehungsweise Zentralfrequenz des Filters an.

Register 23: das MSN gibt die Resonanz im Bereich von 0 bis 15 an. Dadurch werden die Frequenzkomponenten im Bereich der Filterfrequenz besonders hervorgehoben.
Bit 0: leitet Stimme 1 ueber den Filter (falls Bit gesetzt)
Bit 1: leitet Stimme 2 ueber den Filter (falls Bit gesetzt)
Bit 2: leitet Stimme 3 ueber den Filter (falls Bit gesetzt)
Bit 3: leitet externes Signal ueber den Filter (falls Bit gesetzt)

Register 24: LSN: globale Maximallautstaerke von 0 (ausgeschaltet) bis 15 (hoechste Lautstaerke)
Bit 4: LOWPASS des Filters einschalten
Bit 5: BANDPASS des Filters einschalten
Bit 6: HIGHPASS des Filters einschalten
(Kombination ist, wie beschrieben, moeglich)

Bit 7: schaltet Stimme 3 aus, falls Bit gesetzt. Dadurch kann Stimme 3 zur reinen Datengewinnung durch Register 27 und 28 verwendet werden.

Auf all diese Register (#0 bis #24) kann NUR SCHREIBEND zugegriffen werden. Ein Auslesen ist nicht moeglich, da immer der Wert null ausgelesen wird.

Die folgenden vier Register dahingegen koennen nur ausgelesen werden, ein schreibender Zugriff hat keinerlei Auswirkungen:

Register 25: POTX, digitaler Wert des ersten A/D-Wandlers

Register 26: POTY, digitaler Wert des zweiten A/D-Wandlers

Register 27: der Registerinhalt gibt den momentanen Zustand des Oszillators 3 an und dient meist zur Erzeugung von speziellen Effekten.

Register 28: hiermit kann der Verlauf der Huellkurve von Stimme 3 verfolgt werden. Je nach Zustand kann hier ein Wert im Bereich von 0 (niedrigste Lautstaerke) bis 255 (hoechste Lautstaerke) ausgelesen werden.

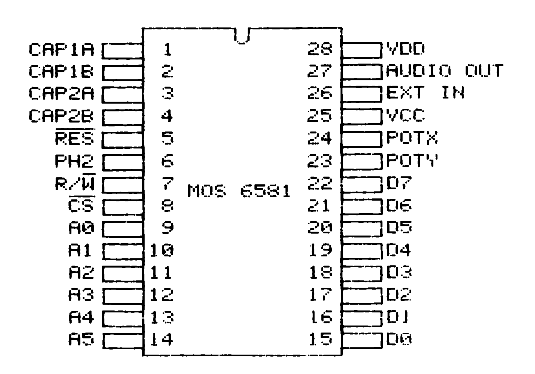
Um einen Ton der gleichschwebend temperierten Skala (zum Beispiel beim Klavier) zu erzeugen, hier noch ein kurzes Beispielprogramm:

```
100 B = 54272 : K = 77 * 2 ↑ 21.25 / 715909
110 IF PEEK (678) THEN K = 495 * 2 ↑ 20.25 / 2216809
120 POKE B + 5, 0 : POKE B + 6, 240 : POKE B + 24, 15
130 FOR I = 0 TO 94 : N = INT (2 ↑ (I / 12) * K + .5)
140 H = INT (N / 256) : POKE B, N - 256 * H : POKE B + 1, H
150 POKE B + 4, 17 : FOR J = 0 TO 99 : NEXT : POKE B + 4, 0
160 NEXT : POKE B + 4, 0
```

Die Pinbelegung des SID (MOS 6581)

Pin	Belegung
Pin 1 - 2:	CAP1A, CAP1B: Anschluss fuer Kondensator, der durch den Filter benoetigt wird. Die Kapazitaet des angeschlossenen Kondensators betraegt beim Commodore 64 2200 pF
Pin 3 - 4:	CAP2A, CAP2B: Kondensatoranschluss fuer Filter; siehe Pin 1 und 2
Pin 5 :	RES/, setzt SID in Grundzustand, alle internen Register werden auf null gesetzt
Pin 6 :	PH2, Systemtaktfrequenz fuer Buskontrolle
Pin 7 :	R/W, Schreib/Leseleitung des Prozessors
Pin 8 :	CS/, Chip-Select, steuert Zugriff auf Register des SID 6581
Pin 9 - 13:	A0 bis A4, Steuerleitungen zur Auswahl der Register des SID, die drei letzten Register sind unbenutzt und haben keine Bedeutung
Pin 14 :	GND, Masseleitung
Pin 15 - 22:	D0 bis D7, Datenbus zum Transfer von Daten vom und zum Prozessor
Pin 23 - 24:	Analogeingaenge (POTY, POTX) fuer Paddles
Pin 25 :	VCC, Versorgungsspannung (+5 Volt)
Pin 26 :	EXT IN, Eingang fuer externes Audio-Signal, das durch den Filter des SID geleitet werden soll
Pin 27 :	AUDIO OUT, Ausgang aller im SID erzeugten Signale sowie EXT IN
Pin 28 :	VDD, Versorgungsspannung (+12 Volt)

PIN CONFIGURATION



Der Commodore 64 verfuegt ueber einen seriellen Bus, an den mehrere Peripheriegeraete angeschlossen werden koennen. Dieser uebertraegt, im Gegensatz zum IEEE-488-Bus der Geraete der CBM-Serie, die Daten seriell (aehnlich der RS-232-Schnittstelle) zwischen Computer und Peripheriegeraet. Die Bedienung des Ports von Maschinensprache aus erfolgt aber ueber Routinen, die von der Funktion her identisch mit denen der CBMs sind. Diese Routinen lassen sind in zwei Gruppen einteilen: Die erste Gruppe bedient die Peripheriegeraete ueber ein Filehandling. Es wird jedem Kanal eine Filenummer zugeordnet (es koennen mehrere Files gleichzeitig geoeffnet - aber nicht aktiv - sein), ein Zugriff erfolgt dann unter Angabe einer Filenummer. Dieses Prinzip entspricht ziemlich genau den BASIC-Befehlen fuer das Arbeiten mit Files: OPEN, CLOSE, PRINT#, INPUT# und GET#. So ist es zum Beispiel durch Aendern der Geraetenummer auch moeglich, eine Ausgabe auf den Bildschirm anstelle einer Ausgabe auf den Drucker zu erreichen.

Die zweite Gruppe von Routinen erfordert ein wesentlich sorgfaeltigeres Arbeiten, da man direkt ueber den seriellen Bus arbeitet. Daher koennen ueber diese Routinen (normalerweise) auch nur Geraetenummern im Bereich von 4 bis 15 verwendet werden. Ein Programm, das ueber diese Routinen zum Beispiel auf die Diskettenstation zugreift, kann nicht (wie beim Arbeiten ueber logische Files) durch Aendern der Geraetenummer fuer Recorder umgeschrieben werden. Auch kann zu einem Zeitpunkt nur ein Kanal geoeffnet sein, durch den auf den seriellen Bus zugegriffen wird.

Hier nun die genaue Vorgehensweise, um einen Kanal fuer den seriellen Bus zu bedienen. Es werden die Kommandos mit den Bezeichnungen versehen, wie sie auch in der Sprungtabelle fuer die Systemroutinen (siehe dort) verwendet worden sind. Um ein Kommando auszufuehren, muss der entsprechende Eintrag in der Sprungtabelle durch "JSR" oder "JMP" aufgerufen werden.

Der erste Befehl dient zur Festlegung, ob das Geraet am seriellen Bus Daten SENDEN oder EMPFANGEN soll. Entsprechend ist das erste Kommando entweder "TALK" oder "LISTEN", der Accu muss beim Aufruf einer dieser Routinen die Geraetenummer enthalten. Die naechste Routine dient zur Ausgabe der Sekundaeradresse, die im Normalfall der Festlegung des Arbeitsmodus' des Geraets dient. Je nachdem, ob die erste Routine "TALK" oder "LISTEN" war, wird als zweite Routine "TKSA" oder "SECOND" verwendet. Der Accu enthaelt hier die Sekundaeradresse, die die Bits 0 bis 3 belegt. Die Bits 4 bis 7 dienen weiterhin der Festlegung, ob ein Kanal (innerhalb des Peripheriegeraets) geoeffnet, geschlossen oder auf einen geoeffneten Kanal zugegriffen werden soll. Beim Schliessen eines Kanals sind die Bits 5 bis 7 gesetzt, beim Zugriff auf einen geoeffneten Kanal sind die Bits 5 und 6 gesetzt, beim Eroeffnen eines Kanals sind alle Bits des MSN gesetzt. Die Ausgabe der Sekundaeradresse beendet automatisch den Kontrollmodus.

Die Ausgabe eines Filenamens kann nach der Neuanmeldung eines Kanals erfolgen, ist jedoch nicht unbedingt erforderlich. Jedes Byte des Filenamens wird dann ueber "CIOUT" (Zeichencode im Accu) zeichenweise ausgegeben. Ist die Ausgabe beendet, so muss die Routine "UNLSN" aufgerufen werden, da die Uebertragung des Filenamens schliesslich abgeschlossen ist. Auf diesen Kanal kann nun unter Angabe der gleichen

Sekundaeradresse wieder zugegriffen werden, was besonders beim Arbeiten mit Diskette wichtig ist. Auch die Ausgabe eines CLOSE-Kommandos auf ein Peripheriegeraets hat - wie das OPEN-Kommando auch - ueber "LISTEN", "SECOND" und "UNLSN" zu erfolgen.

Wurde das Geraet durch gesetztes Bit 5 und Bit 6 in der Sekundaeradresse angesprochen, so koennen nun durch "ACPTR" und "CIOUT" Daten empfangen und gesendet werden. Die Uebergabe der Parameter erfolgt jeweils ueber den Accu. Sind alle Daten uebertragen worden (beim Empfang durch die EOI-Kennzeichnung zu erkennen, jedoch nicht unbedingt bei Geraeten, die nicht von Commodore stammen), wird auch hier der Abschluss des Zugriffs durch "UNLSN" oder "UNTLK" (je nachdem) dem Peripheriegeraet mitgeteilt.

Als Beispiel soll hier das Einlesen des Disketteninhaltsverzeichnisses dienen, das beim Commodore 64 normalerweise ueber 'LOAD "$", 8' und 'LIST' ausgegeben werden muss. Beim folgenden Beispielprogramm wird nun aber nicht mehr das Programm, dass sich im Arbeitsspeicher befindet, ueberschrieben, da das Inhaltsverzeichnis bei Aufruf der Routine direkt ausgegeben wird. Zum besseren Verstaendnis sollte der Aufbau von BASIC-Programmen (Ablage im Arbeitsspeicher) bekannt sein.

```
        LDA #0
        STA STATUS          ;Statusvariable 'ST' loeschen
        LDA #8              ;Geraetenummer fuer Diskettenstation
        JSR LISTEN          ;LISTEN auf Bus ausgeben
        LDA #%11110000      ;Sekundaeradresse fuer OPEN Kanal 0
        JSR SECOND          ;Kanal in Diskettenstation eintragen
        LDA #"$             ;Zeichen fuer Filenamen
        JSR CIOUT           ;Filenamen auf Bus ausgeben
        JSR UNLSN           ;UNLISTEN auf Bus ausgeben
        LDA #8              ;Geraetenummer fuer Diskettenstation
        JSR TALK            ;TALK auf Bus ausgeben
        LDA #%01100000      ;Zugriff auf Kanal 0 der Floppy
        JSR TKSA            ;Sekunaderadresse nach TALK ausgeben
        LDX #5              ;Zaehlwert fuer Schleife
        .BY 44              ;Code fuer BIT-Befehl
START   LDX #3              ;Zaehlwert fuer Schleife
LOOP    JSR GETBYT          ;Zeichen holen, Status pruefen
        DEX
        BNE LOOP            ;Schleife ausfuehren
        TAX                 ;Filelaenge (LSB) in XR uebertragen
        JSR 43735           ;CRLF auf Bildschirm ausgeben
        JSR GETBYT          ;MSB der Anzahl an Bloecken holen
        JSR 48589           ;Ausgabe der Anzahl an Bloecken
LOOP2   JSR GETBYT          ;Zeichen holen, Status pruefen
        TAX                 ;Statusflags setzen
        BEQ START           ;Nullcode? Zeilenende erkannt
        JSR CHROUT          ;Ausgabe Zeichen auf den Bildschirm
        BNE LOOP2           ;Unbedingter Sprung
GETBYT  JSR ACPTR           ;Zeichen von seriellem Bus holen
        LDY STATUS          ;Statusvariable 'ST' pruefen
        BNE STOP            ;Statusbit im Status gesetzt?
        RTS                 ;Nein: Rueckkehr zum Hauptprogramm
STOP    PLA                 ;Ruecksprungadresse vom Stack holen
        PLA
        JSR UNTLK           ;Zugriff auf Kanal beenden
        LDA #8              ;Geraetenummer fuer Diskettenstation
        JSR LISTEN          ;LISTEN auf Bus ausgeben
        LDA #%11100000      ;Sekundaeradresse fuer CLOSE Kanal 0
        JSR SECOND          ;Kanaleintrag loeschen
        JMP UNLSN           ;UNLISTEN auf Bus ausgeben
```

Soll dieses Programm ab der Adresse 828 abgelegt werden, so kann dies durch folgendes Programm geschehen (da das Programm dann im Bereich des Cassettenpuffers liegt, wird es natuerlich durch Recorderoperationen ueberschrieben):

```
100 FOR I = 828 TO 917 : READ A : POKE I, A : NEXT
110 DATA 169, , 133, 144, 169, 8, 32, 177, 255, 169, 240, 32
120 DATA 147, 255, 169, 36, 32, 168, 255, 32, 174, 255, 169
130 DATA 8, 32, 180, 255, 169, 96, 32, 150, 255, 162, 5, 44
140 DATA 162, 3, 32, 124, 3, 202, 208, 250, 170, 32, 215
150 DATA 170, 32, 124, 3, 32, 205, 189, 32, 124, 3, 170, 240
160 DATA 232, 32, 210, 255, 208, 245, 32, 165, 255, 164, 144
170 DATA 208, 1, 96, 104, 104, 32, 171, 255, 169, 8, 32, 177
180 DATA 255, 169, 224, 32, 147, 255, 76, 174, 255
```

Durch Anhaengen der Programmzeile ...

```
190 POKE 874, 202 : POKE 879, 221
```

kann das Programm auf dem VIC-20 betrieben werden (siehe Adressumrechnung fuer VIC-20), da bei diesem das Arbeiten mit dem seriellen Bus in der gleichen Weise erfolgt, wie bei Commodore 64 auch.

DATENTRANSFER UEBER DIE RS-232-SCHNITTSTELLE

Im Betriebssystem des Commodore 64 ist die Geraetenummer zwei der Bedienung der RS-232-Schnittstelle vorbehalten. Dies ist die Bezeichnung fuer einen Standard der seriellen Datenuebertragung und ist aehnlich der europaeischen V.24-Norm. Bei der seriellen Uebertragung wird nicht, wie dies zum Beispiel beim IEEE-488-Bus der Fall ist, ein Byte auf einmal, sondern Bit fuer Bit jede Informationseinheit getrennt uebergeben. Daraus resultiert natuerlich eine langsamere Uebertragungsrate als bei paralleler Uebertragung, dafuer werden aber wesentlich weniger Datenleitungen benoetigt. Ein Anwendungsfall fuer eine serielle Uebertragung waere zum Beispiel der Datenaustausch ueber Telefon.

Zu den bereits im Commodore 64 enthaltenen Routinen fuer die Bedienung der RS-232-Schnittstelle wird nur noch ein Steckmodul benoetigt, das auf den Userport gesteckt wird und die Schnittstelle selbst darstellt, die dann vom Betriebssystem bedient wird. Nun ist es Ihnen mit dem Commodore 64 zum Beispiel auch moeglich, Peripheriegeraete mit RS-232-Schnittstelle zu verwenden.

Wird beim Commodore 64 ein File mit der Geraetenummer zwei eroeffnet, so werden automatisch zwei Pufferbereiche (first in, first out) einer Laenge von je einer Page festgelegt, die fuer den Empfang und fuer das Senden von Daten verwendet werden. Dieser durch den OPEN-Befehl reservierte Bereich von 512 Bytes wird am Ende des BASIC-Arbeitsspeichers angelegt. Nach Anlegen des RS-232-Pufferbereichs wird zusaetzlich ein "CLR" durchgefuehrt, so dass das Oeffnen eines RS-232-Kanals zu Beginn eines Programms durchgefuehrt werden sollte. Es kann immer nur ein RS-232-Kanal auf einmal geoeffnet sein, da durch ein zweites OPEN mit der Geraetenummer zwei die Pointer in die Puffer zurueckgesetzt werden und so bereits empfangene Daten oder noch nicht gesendete Daten verloren gehen. Auch das Schliessen eines RS-232-Kanals, das auch gleichzeitig die 512 Bytes wieder freigibt, beinhaltet ein "CLR".

Achtung! Ist zum Zeitpunkt des Oeffnens des RS-232-Kanals das Programm (ohne Variablen) so lang, dass keine 512 Bytes mehr frei sind, so wird wohl, ohne dass ein Fehler gemeldet werden wird, eine Zerstoerung des Programms die Folge sein.

Saemtliche BASIC-Befehle zur Filebehandlung koennen auch bei Benutzung der RS-232-Schnittstelle Verwendung finden. Das gleichzeitige Empfangen und Senden von Daten ist, sofern spezifiziert, moeglich. Ansonsten geschieht dies nacheinander.

Die Parameter zur Uebertragung werden durch den Filenamen (erstes Byte (Kontrollregister) und zweites Byte (Kommandoregister)) festgelegt. Das dritte und vierte Byte werden nur dann benoetigt, falls eine durch den Benutzer festlegbare Uebertragungsrate spezifiziert werden soll. Wird kein Filename angegeben, so werden die Daten des letzten OPEN-Befehls verwendet.

Das Kontrollregister bestimmt die Uebertragungsrate (Baud-Rate), die Laenge eines Datenworts (fuenf bis acht Bits) und die Anzahl an Stoppbits, die nach der Uebertragung der Datenbits (vor der Uebertragung des Startbits fuer die negative Flanke) gesendet werden.

Durch das Kommandoregister werden der Handshake-Modus und die Art der Uebertragung des Paritybits festgelegt. Ausserdem wird angegeben, ob Vollduplex oder Halbduplex erwuenscht ist.

Die einzelnen Bits des Kontrollregisters sind folgendermassen organisiert: Bit 0 bis Bis 3 geben die Baud-Rate an, Bit 5 und 6 bestimmen die Wortlaenge und Bit 7 ist fuer die Anzahl an Stoppbits zustaendig. Bit 4 ist unbenutzt. Im folgenden eine Uebersicht:

Bit 3	2	1	0	dez	Baud-Rate	
0	0	0	0	0	User Rate	siehe Beschreibung
0	0	0	1	1	50 Baud	
0	0	1	0	2	75 Baud	
0	0	1	1	3	110 Baud	
0	1	0	0	4	134.5 Baud	
0	1	0	1	5	150 Baud	
0	1	1	0	6	300 Baud	
0	1	1	1	7	600 Baud	
1	0	0	0	8	1200 Baud	
1	0	0	1	9	1800 Baud	
1	0	1	0	10	2400 Baud	
1	0	1	1	11	3600 Baud	nicht implementiert
1	1	0	0	12	4800 Baud	nicht implementiert
1	1	0	1	13	7200 Baud	nicht implementiert
1	1	1	0	14	9600 Baud	nicht implementiert
1	1	1	1	15	19200 Baud	nicht implementiert

Bit 6	5	dez	Wortlaenge
0	0	0	8 Bits
0	1	32	7 Bits
1	0	64	6 Bits
1	1	96	5 Bits

Bit 7	dez	Stoppbits
0	0	1 Stoppbit
1	128	2 Stoppbits

Durch das Kommandoregister wird durch Bit 0 der Handshake-Modus festgelegt, Bit 4 legt die Art des Duplex fest und Bit 5 bis 7 sind fuer die Parity-Uebertragung notwendig. Die restlichen Bits (1, 2, 3) sind unbenutzt. Auch hier eine Uebersicht:

Bit 0	dez	Handshake
0	0	3-Line-Handshake
1	1	X-Line-Handshake

Bit 4	dez	Duplex
0	0	Vollduplex
1	16	Halbduplex

Bit 7	6	5	dez	Parity-Uebertragung
-	-	0	0	weder Pruefung, noch Uebertragung
0	0	1	32	ungerade Parity
0	1	1	96	gerade Parity
1	0	1	160	keine Pruefung, Uebertragung Bitwert 1
1	1	1	224	keine Pruefung, Uebertragung Bitwert 0

Hierzu ein praktisches Beispiel:

Es soll ein RS-232-Kanal mit folgenden Parametern eroeffnet werden:

Uebertragungrate:	110 Baud
Wortlaenge:	8 Bits
Anzahl Stoppbits:	2 Stoppbits
Handshake:	3-Line-Interface
Duplex:	Halbduplex
Parity:	Ungerade Parity

Das Eroeffnen koennte nun folgendermassen vorgenommen werden:

```
OPEN 1, 2, 0, CHR$(128+0+3) + CHR$(32+16+0)
```

Die weiteren zwei Zeichen werden nicht benoetigt, da keine eigene Uebertragungsgeschwindigkeit spezifiziert wurde.

Werden Wortlaengen unter acht Bits verwendet, so enthalten unbenutzte Bits den Wert null.

Die Uebernahme von Zeichen sollte durch den GET#-Befehl erfolgen, da bei einem Fehler waehrend einer Uebernahme durch INPUT# (CTS- oder DSR-Missing) sich das System aufhaengt und nur durch RESTORE zurueckgeholt werden kann. Bei Empfang des Null-Zeichens CHR$(0) wird der Leerstring uebernommen.

Bei der Uebergabe eines Wagenruecklaufs an einen Drucker wird nicht gewartet, bis der Drucker wieder bereit ist, Daten zu empfangen (falls der Drucker keine interne Pufferung besitzt). Daher sollte, falls der Drucker diese Moeglichkeit besitzt, X-Line-Interface verwendet werden, da hier die zu uebergebenden Zeichen solange in einem Puffer zwischengespeichert werden, bis der Drucker durch die CTS-Leitung wieder angibt, empfangsbereit zu sein.

Beim Schliessen eines RS-232-Kanals wird jeglicher Datentransfer abgebrochen und saemtliche Ausgaenge werden auf high gesetzt.

Wird der RS-232-Status gelesen, so wird dieser automatisch geloescht, so dass bei mehreren Abfragen der Wert der Statusvariablen einer anderen Variablen uebergeben werden muss. Der RS-232-Status wird nur dann gelesen, falls die letzte Datentransferoperation die RS-232-Schnittstelle betraf. Die Bits des Statusregisters haben bei der Verwendung des RS-232-Kanals folgende Bedeutungen:

Bit	dez	Bedeutung
0	1	Parityfehler
1	2	Framingfehler
2	4	Empfangspuffer ueberlaufen
3	8	Empfangspuffer leer (Zeichen ungueltig)
4	16	CTS (Clear To Send) Signal fehlt
5	32	unbenutzt
6	64	DSR (Data Set Ready) Signal fehlt
7	128	Abbruch erkannt

Ist Bit 2 gesetzt, so sollten mehr als 256 Zeichen im Empfangspuffer zwischengespeichert werden. Der Pufferzeiger wurde zurueckgesetzt. Die Zeichen sind verlorengegangen. Wird versucht ein Zeichen zu lesen, waehrend der Empfangspuffer leer ist, so wird Bit 3 gesetzt.

Die Spezifizierung von eigenen Uebertragungsraten ist moeglich. Hierbei sollte jedoch beachtet werden, dass auf diese Weise nicht die vom Betriebssystem auferlegte obere Grenze von 2400 Baud ueberwunden werden kann, da durch die Taktfrequenz hoehere Geschwindigkeiten nicht mehr verarbeitet werden koennen.

Zum besseren Verstaendnis bei der Errechnung eigener Baud-Raten sollte das Betriebssystemlisting im Bereich von 62499 bis 62540 herangezogen werden. Ausserdem wird die Kenntnis des Sprungvektors in Adresse 65409 (siehe Beschreibung der Systemroutinen) empfohlen.

Ein Programmanfang, der einen RS-232-Kanal mit den im Beispiel spezifizierten Parametern eroeffnet, aber stattdessen eine Uebertragungsrate von 200 Baud (Variable B) definiert, koennte folgendermassen aussehen:

```
100 F=14318180/14:IF PEEK(678) THEN F=17734472/18
110 B=200:T=INT(F/B/2-99.5):H=INT(T/256):L=T-256*H
120 OPEN 1,2,0,CHR$(128+0+0)+CHR$(32+16+0)+CHR$(L)+CHR$(H)
...
```

Allerdings sollte beachtet werden, dass dies keinerlei Standards entspricht. Das obige Programmsegment kann jedoch dann verwendet werden, wenn Sender und Empfaenger auf die Baud-Raten eingerichtet sind.

Zum Abschluss noch eine Tabelle der beim RS-232-Handling verwendeten, RS-232-spezifischen Adressen in der Zeropage, Page 2 und im I/O-Bereich der NMI-CIA:

```
167     : Zwischenspeicher fuer empfangenes Bit
168     : Bitzaehler fuer den Empfang von Bytes
169     : Flag fuer Empfang des Startbits
170     : serielles Shift-Register fuer Empfang von Daten
171     : Register zur Errechnung der Parity beim Empfang

180     : Bitzaehler fuer die Ausgabe von Bytes
181     : naechstes zu sendendes Bit (Bit 2)
182     : serielles Shift-Register fuer Ausgabe von Daten
189     : Register zur Bestimmung der Parity bei der Ausgabe

247/248: Zeiger auf Beginn des Empfangspuffers;
         gueltig, falls Highbyte ungleich null
249/250: Zeiger auf Beginn des Sendepuffers;
         gueltig, falls Highbyte ungleich null

659     : Kontrollregister (siehe Beschreibung)
660     : Kommandoregister (siehe Beschreibung)
661/662: Wert fuer Baud-Rate aus Tabelle
663     : RS-232-Statusbyte
664     : Wortlaenge (Berechnung bei OPEN)
665/666: Wert fuer Timer beim Senden
667     : Zeiger auf Ende des Empfangspuffers
          (letztes Zeichen + 1)
668     : Zeiger auf Zeichen im RS-232-Empfangspuffer
          (naechstes Zeichen bei GET)
669     : Zeiger auf zu uebertragendes Byte
670     : Zeiger auf naechste freie Stelle im Sendepuffer
673     : Flagregister fuer aktive NMIs (Datenuebertragung)
          Bit 0 ist beim Senden von Daten gesetzt; Bit 1 und
          Bit 4 werden beim Empfang benoetigt, Bit 1 fuer
          Timeout-TimerB und Bit 4 fuer die Erkennung des
          Startbits beim Uebergang vom high des Stoppbits zum
          low des Startbits.
```

Verwendete Adressen der NMI-CIA:

56576	: Port A, siehe unten
56577	: Port B, siehe unten
56580/56581	: TimerA, Baud-Rate fuer Ausgabe von Daten
56582/56583	: TimerB, Baud-Rate fuer Empfang von Daten
56589	: ICR, Festsetzung der aktiven NMIs, Feststellung der NMI-Quelle
56590	: CRA, Festlegung des TimerA Arbeitsmodus'
56591	: CRB, Festlegung des TimerB Arbeitsmodus'

Belegung des User-Ports in Verbindung mit den CIA-Ports:

Pin	6526	Beschreibung	Richtung
A	-	GND : Protective Ground	-
B	FLAG	Sin : Received Data (fuer Startbit)	IN
C	PB0	Sin : Received Data, RS-232 IN	IN
D	PB1	RTS : Request To Send	OUT
E	PB2	DTR : Data Terminal Ready	OUT
F	PB3	RI : Ring Indicator	IN
H	PB4	DCD : Data Carrier Detect	IN
J	PB5	unbenutzt	IN
K	PB6	CTS : Clear To Send	IN
L	PB7	DSR : Data Set Ready	IN
M	PA2	Sout: Transmitted Data, RS-232 OUT	OUT
N	-	GND : Signal Ground	-

DER I/O BAUSTEIN MOS 6526 (CIA) DES COMMODORE 64

Zwei I/O-Bausteine des Typs 6526 besitzt der Commodore 64. Durch sie wird der Kontakt vom Computer zur Aussenwelt geschaffen. Die Tastaturabfrage, die Erkennung von Signalen auf Cassette, die Kommunikation ueber den seriellen Bus oder die RS-232-Schnittstelle, all dies und noch einiges mehr ist erst durch diese beiden Bausteine moeglich, deren Funktionen hier dargelegt werden sollen.

Zuerst eine allgemeine Beschreibung, die fuer beide CIAs (Complex Interface Adapter) gilt. Auf die Unterschiede zwischen den beiden Bausteinen, die nur in deren Verwendung beim Commodore 64 liegen, wird im Anschluss eingegangen.

Eine kurze Uebersicht: Dieser Baustein verfuegt ueber 16 einzeln programmierbare Portleitungen sowie ueber zugehoerige Kontrolleitungen, die fuer Handshaking verwendet werden koennen, zwei kombinierbare 16-Bit-Timer und ein 8-Bit-Shiftregister fuer serielle Datenuebertragungen. Ausserdem existiert eine interne 24-Stunden-Uhr mit programmierbarer Alarmzeit.

Die Steuerung all dieser Moeglichkeiten erfolgt ueber sechzehn Register, die verschiedene Funktionen erfuellen. Vier dieser Register dienen der Programmierung der ...

PORTS DER CIA

Die CIA besitzt zwei Ports zu je einer Breite von acht Bits. Jede Portleitung der beiden Ports (Port A und Port B genannt) kann unabhaengig als Eingang oder Ausgang geschaltet werden. Dies geschieht ueber die Datenrichtungsregister der CIA. Entsprechend den beiden Ports existieren zwei Datenrichtungsregister mit den Bezeichnungen "DDRA" und "DDRB" (Data Direction Register). Jedes Bit eines Datenrichtungsregisters gehoert entsprechend zu einem Bit eines der beiden Register, in denen dann der Zustand der Portleitung geaendert oder geprueft werden kann. Diese beiden Register heissen "PRA" und "PRB" (Peripheral Data Register).

Um eine Portleitung als EINGANG zu kennzeichnen, muss das zugehoerige Bit im Datenrichtungsregister GELOESCHT werden. Entsprechend ist das Bit im Datenrichtungsregister zu SETZEN, wenn die Portleitung als AUSGANG verwendet werden soll.

Die Datenrichtungsregister befinden sich an den Adressen 2 (DDRA) und 3 (DDRB) relativ zur Basisadresse des Bausteins. Die Ports nehmen die Adressen 0 (PRA) und 1 (PRB) ein. Die Nummern der Portleitungen (PA0 bis PA7, PB0 bis PB7) entsprechen den Nummern der Bits der zugehoerigen Steuerregister. PB6 und PB7 werden auch im Zusammenhang mit den Timern verwendet.

Die Uebergabe von Daten zwischen zwei CIAs (oder aehnlichen Bausteinen) kann unter Kontrolle der Leitungen PC und FLAG erfolgen. So wird PC fuer die Dauer eines Taktes auf Low-Pegel gelegt, wenn auf PRB lesend oder schreibend zugegriffen wurde. Dieses Signal kann verwendet werden, um zu zeigen, dass die Daten (die von anderer Stelle auf den Port B gegeben wurden) vom Computer gelesen worden (DATA ACCEPTED) sind. Andererseits kann dadurch auch erkannt werden, dass neue Daten vorliegen und gelesen werden koennen (DATA VALID). Dies kann durch die FLAG-Leitung festgestellt werden. Sie erkennt negative Flanken (Wechsel von high nach low) und gibt diese Information in Form einer gesetzen Flag

in einem Register der CIA weiter. Ausserdem kann zusaetzlich ein Interrupt ausgeloest werden. Sollen also Daten zwischen zwei Computern durch deren CIAs ausgetauscht werden, so genuegt es, die PC-Leitung der einen CIA mit der FLAG-Leitung der anderen CIA (und umgekehrt) zu verbinden (natuerlich zusaetzlich zu Portleitungen von Port B).

DER SERIELLE PORT

Der Serielle Port besteht aus einem Shiftregister, das mit Daten einer Breite von acht Bits arbeitet. Dieses Shiftregister kann sowohl als Eingang, als auch als Ausgang programmiert werden. Dies geschieht durch Bit 6 von CRA. Ist dieses Bit geloescht, so fungiert der serielle Port als Eingang, ansonsten als Ausgang (gleiche Spezifikation, wie bei den Datenrichtungsregistern auch). Als Leitungen fuer die Uebergabe von Daten durch den seriellen Port dienen die Pins SP und CNT. Dabei werden die Daten auf SP uebertragen, die Steuerung erfolgt durch CNT. Ist das Shiftregister auf Eingang geschaltet, so wird im Falle einer positiven Flanke auf CNT der am Eingang SP befindliche Wert in das Shiftregister geschoben. Ist dies achtmal geschehen, so wird der Wert des Shiftregisters in das Serial Data Register (SDR) uebertragen. Gleichzeitig wird (sofern zugelassen) ein Interrupt generiert, woraufhin dann die gelesenen Daten durch den Computer verarbeitet werden koennen. Im Ausgabemodus wird die Geschwindigkeit, mit der die Daten aus dem Shiftregister auf den seriellen Port ausgegeben werden, durch Timer A spezifiziert. Dabei betraegt die Baudrate die halbe Underflow-Rate von Timer A. Die hoechste Uebertragungsrate waere demnach ein Viertel des Systemtakts, was jedoch in der Praxis kaum erreicht werden kann, da die Daten schliesslich vom Empfaenger auch noch verarbeitet werden muessen. Sind Daten ins SDR geschrieben worden, so beginnt die Uebertragung der Daten (sofern Timer A laeuft und sich im Continuous Mode befindet). Am CNT-Pin erscheint nun die aus Timer A abgeleitete Frequenz, die Daten werden aus dem Shiftregister (auf SP) geschoben, waehrend eine positive Flanke auf CNT ausgegeben wird. Dabei wird das MSB des seriellen Ports zuerst herausgeschoben (eingehende Daten sollten das gleiche Format aufweisen, da sie ansonsten erst umgewandelt werden muessen). Sind alle acht Bits uebertragen worden, wird (ebenso wie beim Empfang von Daten) ein Interrupt ausgeloest, der dem Programm mitteilt, dass weitere Daten uebertragen werden koennen. Dies geschieht nicht, wenn bereits vor der Uebertragung des achten Bits neue Daten ins SDR geschrieben wurden. In diesem Fall wuerde die Uebertragung direkt fortgesetzt.

DIE TIMER

Jeder der beiden Timer der CIA besteht aus einem Vorspeicher (Latch) und Zaehler (Counter) mit einer Breite von je sechzehn Bits. Wird in die zu einem Timer gehoerigen Register GESCHRIEBEN, so wird dieser Wert automatisch ins Latch uebertragen. Beim LESEN dagegen erscheint immer der momentane Zaehlerwert. Die beiden Timer koennen unabhaengig voneinander benutzt werden, es besteht jedoch auch die Moeglichkeit, sie miteinander zu verbinden. Weiterhin bestehen mehrere verschiedene Modi, die es erlauben, lange Verzoegerungsschleifen zu programmieren oder zum Beispiel spezielle Wellenformen mit unterschiedlich langen Impulslaengen zu generieren. Das Lesen des Counters und Schreiben des Latch erfolgt fuer Timer A ueber die Register 4 (low) und 5 (high), Timer B wird ueber die Register 6 und 7 (ebenfalls low und high) programmiert.

Die Kontrolle des Arbeitsmodus' der beiden Timer erfolgt ueber ihnen zugeordnete (voneinander unabhaengige) Kontrollregister (CRA und CRB, Register 14 und 15), deren Bits folgende Bedeutungen haben:

Bit 0: START/STOP

Durch das Loeschen dieses Bit kann ein Timer jederzeit angehalten oder durch das Setzen dieses Bits gestartet werden.

Bit 1: PB ON/OFF

Sollen Signale unter Kontrolle eines Timers an einem Port erzeugt werden, so kann dies durch dieses Bit gesteuert werden. Bei jedem Timeout wird der zum Timer gehoerige Port (PB6 fuer Timer A, PB7 fuer Timer B) entsprechend den Angaben in Bit 2 beeinflusst (sofern Bit 1 gesetzt ist! Ist es geloescht, so hat der Timer keinerlei Einfluss auf die Portleitung). Diese Funktion hat eine hoehere Prioritaet als die Funktionszuweisung von PB6 und PB7 im DDRB. Ist dieses Bit also gesetzt, so fungiert dieser Port ohne Ruecksicht auf den Wert im DDRB als Ausgang.

Bit 2: TOGGLE/PULSE

Dieses Bit gibt an, in welcher Weise die zum Timer gehoerige Portleitung beim Timeout des Timers arbeiten soll. Ist das Bit geloescht, so wird bei jedem Timeout auf der zugehoerigen Portleitung ein positives Signal der Laenge eines Taktzyklus gegeben. Die zweite Moeglichkeit ist, durch diesen Port Rechtecksignale bestimmter Frequenz auszugeben. Ist dieses Bit naemlich gesetzt, so wird bei jedem Timer-Underflow das Signal an diesem Port invertiert. Die Frequenz dieses Signals entspricht dadurch (wie beim Arbeiten mit dem Seriellen Port auch) der halben Underflow-Rate des Timers. Beide Modi jedoch arbeiten nur dann, wenn Bit 1 auch gesetzt ist.

Dieser Modus ist uebrigens identisch mit dem des "OUTPUT ENABLE" durch Setzen von Bit 7 im ACR der VIA 6522, wodurch ueber PB7 Rechtecksignale ausgegeben werden koennen, deren Frequenz durch Timer 1 festgelegt wurde.

Bit 3: ONE-SHOT/CONTINUOUS

Im One-Shot-Modus (Bit gesetzt) zaehlt der Counter vom gespeicherten Wert bis auf den Wert '0' herunter, erzeugt einen Interrupt, laedt den Timer neu mit dem im Latch befindlichen Wert und stoppt dann. Im Continuous Mode stoppt der Timer nicht, sondern beginnt dann diese Prozedur von vorne.

Bit 4: FORCE LOAD

Wird in dieses Bit der Wert '1' geschrieben, so wird der Counter des Timers, unabhaengig davon, ob er gerade laeuft oder nicht, mit dem im Latch befindlichen Wert geladen. Beim Lesen ist dieses Bit immer geloescht. Das Schreiben einer '0' hat keinerlei Wirkung.

Ein Counter wird ausserdem automatisch immer dann mit dem Wert des Latch geladen, falls ein Underflow auftritt oder (nur wenn der Timer momentan gestoppt ist) wenn in das Highbyte des Timers geschrieben wird. Werden also neue Latchwerte gespeichert, so braucht kein FORCE LOAD zu erfolgen, wenn diese Werte in der Reihenfolge low/high gespeichert werden.

Die folgenden Bits sind fuer CRA und CRB unterschiedlich belegt und bieten fuer die verschiedenen Timer verschiedene Moeglichkeiten, welches die Quelle fuer das Herabzaehlen des Timers sein soll.

CRA, Bit 5: IN MODE

Ist dieses Bit geloescht, so wird Timer A mit der Frequenz des Systemtakts herabgezaehlt. Bei gesetztem Bit 5 wird Timer A bei jedem positiven Impuls auf CNT um den Wert '1' vermindert.

CRB, Bit 5 und Bit 6: IN MODE

Fuer Timer B existieren zu den beiden Modi von Timer A (die fuer Timer B bei geloeschtem Bit 6 Gueltigkeit haben) noch zwei weitere Taktquellen: Bei der ersten (Bit 6 gesetzt, Bit 5 geloescht) wird Timer B bei jedem Underflow von Timer A herabgezaehlt. Dadurch ist es moeglich, beide Timer zu einem Timer mit einer Breite von 32 Bits zu kombinieren. Als weiterer Arbeitsmodus bietet sich die Moeglichkeit, Timer B nur dann die Underflow-Impulse von Timer A zaehlen zu lassen, wenn waehrenddessen CNT auf High-Pegel liegt.

TIME OF DAY CLOCK

Die CIA verfuegt ueber eine 24-Stunden-Echtzeituhr mit einer Aufloesung von einer Zehntelsekunde. Zusaetzlich besteht die Moeglichkeit, eine Alarmzeit festzulegen. Ist diese Alarmzeit durch die Uhr erreicht worden, so wird ein Interrupt ausgeloest. Die Uhr ist in vier Register aufgeteilt: ein Stundenregister, dessen MSB die Flag fuer AM/PM darstellt, ein Minuten- und Sekundenregister sowie ein weiteres Register fuer die Zehntelsekunden. Die Zahlendarstellung in all diesen Registern erfolgt im gepackten BCD-Format, wodurch eine Umwandlung der Daten in den Registern durch Maschinenprogrammen vereinfacht (und beschleunigt) wird.

Die Uhr wird ueber die Netzfrequenz getaktet, wodurch eine recht hohe Genauigkeit gewaehrleistet ist. Es werden Netzfrequenzen von 50 und 60 Hertz (einstellbar) verarbeitet. Die Einstellung erfolgt durch Bit 7 des CRA (geloescht = 60 Hz, gesetzt = 50 Hz).

Die Alarmzeit wird durch die gleichen Register wie die normale Zeit auch festgesetzt. Die Unterscheidung erfolgt durch Bit 7 des CRB. Ist es gesetzt, so werden die in die Register 8 bis 11 (Zehntelsekunden bis Stunden) geschriebenen Daten als Alarmzeit aufgefasst. Ist es geloescht, so kann die normale Uhrzeit festgelegt werden. Das Lesen der Register gibt immer die Uhrzeit an. Die Alarmzeit kann daher nicht gelesen werden.

Soll die Uhrzeit gelesen werden, so werden beim Lesen des Stundenregisters automatisch saemtliche Register fuer die Uhrzeit in ein Latch uebertragen, sodass waehrend des Lesens keinerlei Uebertrag auftreten kann. Es kann nun also die Uhrzeit zum Zeitpunkt, zu dem das Stundenregister gelesen wurde, verarbeitet werden - es treten keinerlei Uebertragsprobleme mehr auf. Die Uhr "steht" nun scheinbar, laeuft intern jedoch weiter. Ist zum Abschluss das Register fuer die Zehntelsekunden gelesen worden, so erscheint die Uhrzeit nun wieder direkt in den Registern, es erfolgt keinerlei Zwischenspeicherung mehr. Natuerlich ist es moeglich, einzelne Register zu lesen, ohne dass eine Zwischenspeicherung erfolgt. Schliesslich tritt beim Lesen von einzelnen Registern

nicht mehr das Problem des Uebertrags auf. Soll jedoch das Stundenregister alleine gelesen werden, so muss anschliessend das Register fuer Zehntelsekunden gelesen werden, um das Latchen wieder aufzuheben.

Aehnlich erfolgt dies beim Setzen der Uhrzeit. Sobald in das Stundenregister geschrieben wurde, wird die Uhr sofort gestoppt. Sie laeuft erst dann wieder an, wenn das Register fuer Zehntelsekunden gesetzt wurde. Dadurch wird erreicht, dass die Uhr auch wirklich erst zu dem Zeitpunkt anfaengt zu laufen, der durch die neuen Werte festgelegt wurde.

INTERRUPT HANDLING

Bei einer CIA existieren fuenf Moeglichkeiten des Interrupts, die einzeln kontrolliert werden koennen. Dazu dient das Register 13. Durch dieses wird festgelegt, ob durch eine Interruptquelle auch die Interruptleitung des Prozessors auf low gelegt werden soll. Es ist also moeglich, jeden einzelnen Interrupt getrennt zu sperren oder freizugeben. Ausserdem kann, unabhaengig davon, ob der Interrupt zugelassen ist oder nicht, durch Lesen von Register 13 festgestellt werden, ob das geforderte Ereignis fuer einen Interrupt aufgetreten ist. Es kann jedoch nicht der Zustand des Interrupt Enable Registers festgestellt werden. Daher kann es notwendig sein, ein weiteres Register im RAM zu verwenden, um aktive Interrupts zu merken (dies wird beim RS-232-Handling gemacht).

Hier die Belegung des Registers:

Bit 0: Underflow von TIMER A
Bit 1: Underflow von TIMER B
Bit 2: TIME OF DAY CLOCK = ALARM
Bit 3: Shiftregister voll/leer (je nach Modus)
Bit 4: negative Flanke auf FLAG

Beim Lesen von Register 13 sind im gelesenen Wert nun saemtliche Bits gesetzt, fuer die die Bedingung eines Interrupts erfuellt ist (Bit 5 bis Bit 7 sind unbenutzt). Beim Lesen dieses Registers werden ausserdem saemtliche Bits rueckgesetzt, sodass der gelesene Wert bei mehrfacher Verwendung zwischengespeichert werden muss.

Soll ein Interrupt (oder mehrere) GESPERRT werden, so muss ein Wert in Register 13 geschrieben werden, in dem saemtliche Bits gesetzt sind, deren entsprechende Interruptquellen gesperrt werden sollen. Soll ein Interrupt FREIGEGEBEN werden, so muss zusaetzlich Bit 7 in diesem Wert gesetzt sein. Bit 7 fungiert also als Flag dafuer, ob ein Interrupt gesperrt oder freigegeben werden soll.

Registeruebersicht:

Register 0: PRA (Port Register A)
Dient zur Feststellung der Zustaende von Port A (Eingang) und zum Aendern derselben (Ausgang).

Register 1: PRB (Port Register B)
Dient zur Feststellung der Zustaende von Port B (Eingang) und zum Aendern derselben (Ausgang).

Register 2: DDRA (Data Direction Register A)
Festlegung der einzelnen Portleitungen von Port A als Eingang (Bit geloescht) oder Ausgang (Bit gesetzt).

Register 3: DDRB (Data Direction Register B)
Festlegung der einzelnen Portleitungen von Port B als Eingang (Bit geloescht) oder Ausgang (Bit gesetzt).

Register 4: TA LO (Timer A Low Order Byte)
Beim LESEN dieses Registers wird der momentane Wert des Counters (LSB) von Timer A zurueckgegeben.
Ein SCHREIBENder Zugriff setzt das LSB des Latch von Timer A fest.

Register 5: TA HI (Timer A High Order Byte)
Beim LESEN dieses Registers wird der momentane Wert des Counters (MSB) von Timer A zurueckgegeben.
Ein SCHREIBENDER Zugriff setzt das MSB des Latch von Timer A fest. Ausserdem wird, sofern Timer A gestoppt ist, der Wert des Latch in den Counter uebertragen.

Register 6: TB LO (Timer B Low Order Byte)
Die Funktion dieses Registers entspricht der des Registers 4, jedoch fuer Timer B.

Register 7: TB HI (Timer B High Order Byte)
Die Funktion dieses Registers entspricht der des Registers 5, jedoch fuer Timer B.

Register 8: TOD 10THS (10ths of Seconds Register)
LESEN: Zehntelsekunden der internen Uhr im BCD-Format, gleichzeitiges Aufheben einer durch das Lesen des Stundenregisters hervorgerufenen Zwischenspeicherung.
SCREIBEN: Festlegung der Zehntelsekunden der internen Uhr sowie gleichzeitiges Starten der Uhr (falls MSB von CRB gesetzt), ansonsten Festsetzung der Zehntelsekunden der Alarmzeit.

Bit 4 bis Bit 7 sind unbenutzt.

Register 9: TOD SEC (Seconds Register)
LESEN: Sekunden der internen Uhr im BCD Format.
SCREIBEN: Festlegung der Sekunden der internen Uhr (falls MSB von CRB gesetzt), ansonsten Festsetzung der Sekunden der Alarmzeit.

Bit 0 bis Bit 3 enthalten die Einerstelle, Bit 4 bis 6 die Zehnerstelle der Sekunden. Bit 7 ist unbenutzt.

Register 10: TOD MIN (Minutes Register)
Funktion und Belegung entsprechend Register 9, jedoch fuer die Minuten.

Register 11: TOD HR (Hours + AM/PM Register)
Bit 0 bis Bit 3: Einerstelle der Stunden
Bit 4 : Zehnerstelle der Stunden
Bit 5 und Bit 6: unbenutzt
Bit 7 : AM (=0) und PM (=1) Flag
LESEN: Stunden und AM/PM-Flag der internen Uhr, gleichzeitig wird die Uhrzeit in ein internes Latch uebertragen, damit es nicht zu Ueberlaufproblemen kommt. Die Zwischenspeicherung wird durch Lesen von Register 8 aufgehoben.
SCREIBEN: Festlegung der Stunden (und AM/PM) der internen Uhr sowie gleichzeitiges Stoppen der Uhr (falls MSB von CRB gesetzt), ansonsten Festsetzung der Stunden (und AM/PM) der Alarmzeit.

Register 12: SDR (Serial Data Register)
Dieses Register dient dazu, die Daten fuer den Seriellen Port festzulegen (WRITE) oder gelesene Daten zu uebernehmen (READ).

Register 13: ICR (Interrupt Control Register)
LESEN (Interrupt DATA): Feststellung, ob Interrupts aufgetreten sind (Bit gesetzt). Bit 7 ist gesetzt, wenn mindestens ein Interrupt freigegeben und die Bedingung fuer diesen Interrupt gegeben ist. Saemtliche Bits werden geloescht, wenn dieses Register gelesen wurde.
SCHREIBEN (Interrupt MASK): Jegliche Interrupts, deren zugehoerige Bits gesetzt sind, werden, falls Bit 7 des geschriebenen Wertes gleich eins ist (MASK Bit wird gesetzt), freigegeben. Ist Bit 7 geloescht, so wird dieser Interrupt gesperrt (MASK Bit wird geloescht). Die Interrupts, deren korrespondierende Bits im geschriebenen Wert geloescht sind, bleiben unbeeinflusst.

Bit 0: Underflow von Timer A
Bit 1: Underflow von Timer B
Bit 2: interne Uhr hat Alarmzeit erreicht
Bit 3: Shiftregister voll/leer (Input/Output)
Bit 4: negative Flanke auf FLAG
Bit 5 und Bit 6 sind unbenutzt.

Register 14: CRA (Control Register A)
Bit 0: START
Ist dieses Bit gesetzt, so laeuft Timer A momentan, ansonsten ist er gestoppt. Durch Setzen dieses Bits kann Timer A gestartet werden, beim Loeschen wird Timer A sofort angehalten.
Bit 1: PB ON
Ist dieses Bit gesetzt, so wird ein Underflow von Timer A auf die im naechsten Bit gekennzeichnete Weise auf PB6 sichtbar gemacht. Ist Bit 1 geloescht, so hat Timer A keinerlei Einfluss auf PB6.
Bit 2: OUT MODE
Bei gesetztem Bit (TOGGLE) wird bei jedem Underflow von Timer A das Signal auf PB6 invertiert. Ist es geloescht (PULSE), so wird bei einem Underflow ein positiver Impuls fuer die Dauer eines Systemtakts ausgegeben.

Bit 3: RUN MODE
Durch Setzen dieses Bits wird bewirkt, dass Timer A, nachdem er vom gelatchten Wert auf null zurueckgezaehlt hat, anhaelt (Counter wird neu geladen). Ist es geloescht, so wird fortlaufend vom Startwert auf null zurueckgezaehlt.

Bit 4: LOAD
Durch Setzen dieses Bit wird ein Laden des Counters mit dem im Latch gespeicherten Wert erzwungen. Beim Lesen ist dieses Bit jedoch immer geloescht.

Bit 5: IN MODE
Timer A zaehlt die Impulse des Systemtakts, wenn dieses Bit geloescht ist, bei gesetztem Bit zaehlt Timer A die positiven Impulse auf CNT.

Bit 6: SP MODE
Dieses Bit schaltet den Seriellen Port auf Eingang (=0) oder Ausgang (=1).

Bit 7: TOD IN
Die Einstellung der zu verarbeitenden Netzfrequenz fuer die interne Uhr erfolgt ueber dieses Bit. 50 Hz werden verarbeitet, wenn dieses Bit gesetzt ist, ansonsten werden 60 Hz am TOD-Pin erwartet.

Register 15: CRB (Control Register B)
Die Bits 0 bis 4 entsprechen in der Funktion denen des Registers CRA, jedoch fuer Timer B und PB7.

Bit 5 und Bit 6: IN MODE
Ist Bit 6 geloescht, so entspricht der Zaehlmodus von Timer B der in Bit 5 festgelegten Weise (entsprechend der Bedeutung von IN MODE in CRA). Ist Bit 6 gesetzt, so zaehlt Timer B die Underflows von Timer A (Bit 5 geloescht) oder nur dann die Underflows von Timer A, wenn CNT auf high liegt (Bit 5 gesetzt).

Bit 7: ALARM
Das Setzen dieses Bits bewirkt, dass beim schreibenden Zugriff auf die Register 8 bis 11 nicht die normale Uhrzeit (wie dies der Fall ist, wenn dieses Bit geloescht ist) gesetzt wird, sondern die Alarmzeit, die jedoch nicht ausgelesen werden kann.

DIE VERWENDUNG DER CIAs IM COMMODORE 64

Die Register der beiden CIAs belegen im Commodore 64 den Bereich von 56320 bis 56335 (CIA #1) sowie den Bereich von 56576 bis 56591 (CIA #2). Einer der Hauptunterschiede zwischen den beiden CIAs besteht darin, dass die Interrupt-Leitung der CIA #1 mit der IRQ-Leitung des Prozessors verbunden ist. Ist also eine der Interruptquellen der CIA freigegeben, so kann sie beim Prozessor einen IRQ ausloesen. Dahingegen ist die zweite CIA mit der NMI-Leitung verbunden. Sie wird zum Ausloesen der Interrupts fuer das RS-232-Handling verwendet, da dieses (aus Geschwindigkeitsgruenden) "parallel" zu allen anderen Aktivitaeten des Prozessors abgewickelt werden muss und daher die hoechste Prioritaet bekommt. Daher werden die beiden CIAs auch nach diesen Verbindungen zur CPU bezeichnet: IRQ-CIA fuer CIA #1, NMI-CIA fuer CIA #2. Die Register sind in der normalen Reihenfolge von 0 bis 15 angeordnet.

Hier die genaue Verwendung der einzelnen Moeglichkeiten der beiden CIAs im Commodore 64:

IRQ-CIA (CIA #1):

Die beiden Ports dienen der Abfrage der Joysticks, Tastatur und der Auswahl der Paddles:

Port A: Jedes der acht Bits kann eine Reihe der Tastatur zur Dekodierung anwaehlen, wenn das entsprechende Bit geloescht ist.
Bit 0 bis Bit 4 dienen ausserdem der Abfrage von Controlport 2, Bit 6 und Bit 7 der Auswahl des Paddlepaares (siehe Erklaerung von Paddle und Joystick).

Port B: Dieser Port dient der Spaltenrueckmeldung der ausgewaehlten Tastaturreihe fuer die Tastaturdekodierung. Eine Taste der ausgewaehlten Reihe ist gedrueckt, wenn das entsprechende Bit geloescht ist. Bits 0 bis 4 werden auch hier zur Joystickabfrage (Controlport 1) verwendet.

Timer A wird zur Erzeugung des "normalen" IRQs alle 60stel Sekunden benutzt. Ist ein Underflow aufgetreten, so wird unter anderem die Tastatur abgefragt, die softwaremaessig realisierte Uhr (nicht die CIA-Uhr!) weitergestellt und anderes erledigt. Ausserdem wird dieser Timer fuer das Lesen von Cassette benoetigt.

Timer B wird nur bei Cassettenoperationen (beim Schreiben zur Festlegung der Impulslaengen, beim Lesen zur Feststellung der Impulsabstaende) sowie bei der Benutzung des seriellen Bus' (fuer Wartezeiten) verwendet und sollte daher fuer eigene Programme problemlos verwendet werden koennen.

Die eingebaute Echtzeituhr wird (zusammen mit Timer A) vom Betriebssystem nur gelesen, um daraus Zufallszahlen bei "RND(0)" zu generieren. Dies sollte jedoch andere Programme, die die Echtzeituhr benutzen, nicht beeinflussen.

Der Serielle Port ist, wie das Pin CNT auch, vollkommen unbenutzt und ist fuer eigene Anwendungen an den User-Port herausgefuehrt.

Der FLAG-Eingang wird zum Lesen von Cassette und als SRQ-Eingang (Service ReQuest) am seriellen Bus verwendet.

NMI-CIA (CIA #2):

Die beiden Ports werden hauptsaechlich fuer Datenuebertragung genutzt:

Port A: Bit 0 und Bit 1 dienen zur Festlegung der Video-Bank fuer den VIC-II-Chip (hoechstwertige Adressbits VA14 und VA15). Bit 2 wird fuer die Datenausgabe der RS-232-Uebertragung verwendet. Die uebrigen Bits dienen dem Austausch von Daten ueber den seriellen Bus:

Bit 3: ATN OUT (Attention-Leitung)
Bit 4: CLOCK OUT (Taktleitung Ausgang)
Bit 5: SERIAL OUT (Datenleitung Ausgang)
Bit 6: CLOCK IN (Taktleitung Eingang)
Bit 7: SERIAL IN (Datenleitung Eingang)

Port B ist vollstaendig an den Userport herausgefuehrt und im Normalfall unbenutzt. Allerdings erhalten die Portleitungen von Port B (mit Ausnahme von PB5) eine Bedeutung im Zu-

sammenhang mit der RS-232-Schnittstelle, die auf den Userport aufgesteckt wird. Die Bedeutungen der Leitungen sind im Zusammenhang mit der Erklaerung des RS-232-Handlings erlaeutert.

Timer A wird beim Senden von Daten ueber die RS-232-Schnittstelle verwendet und ist ansonsten unbenutzt.

Timer B wird beim Empfangen von Daten ueber die RS-232-Schnittstelle verwendet und ist ansonsten unbenutzt.

Die eingebaute Echtzeituhr wird in keiner Weise vom Betriebssystem angesprochen und steht vollkommen frei fuer eigene Anwendungen.

Wie bei der IRQ-CIA auch, so ist auch hier der Serielle Port vollkommen unbenutzt und ist fuer eigene Anwendungen an den User-Port (SP und CNT) herausgefuehrt.

FLAG und PC zum Uebertragen von Daten mittels Handshaking sind (ausser beim RS-232-Handling, hier wird FLAG zur Erkennung des Startbits herangezogen) unbenutzt und stehen am Userport frei zur Verfuegung.

PINBELEGUNG DER CIA

```
Pin  1      : GND, Masseleitung
Pin  2 -  9: bidirektionale Portleitungen PA0 bis PA7
Pin 10 - 17: bidirektionale Portleitungen PB0 bis PB7
Pin 18      : PC/, Handshakeleitung fuer Port B, Ausgang
Pin 19      : TOD, Eingang Netzfrequenz fuer "Time of Day"
Pin 20      : VCC, Betriebsspannung von +5 Volt
Pin 21      : IRQ/, Interruptleitung zum Prozessor (NMI, IRQ)
Pin 22      : R/W, Eingang Schreib/Leseleitung des Prozessors
Pin 23      : CS/, Kennzeichnung Zugriff auf CIA-Register
Pin 24      : FLAG/, Handshakeleitung fuer Port B, Eingang
Pin 25      : PH2, Systemtakt 2 fuer Prozessorbussteuerung
Pin 26 - 33: DB7 bis DB0, Datenbus des Prozessors
Pin 34      : RES/, Ruecksetzen der CIA in Ausgangszustand
Pin 35 - 38: RS3 bis RS0, Registerauswahl bei CS/=0
Pin 39      : SP serieller Port, Eingang oder Ausgang von SDR
Pin 40      : CNT, Steuerleitung zu SDR oder fuer Timer
```

MOS 6526

Signal	Pin	Pin	Signal
GND	1	40	CNT
PA0	2	39	SP
PA1	3	38	RS0
PA2	4	37	RS1
PA3	5	36	RS2
PA4	6	35	RS3
PA5	7	34	$\overline{RES}$
PA6	8	33	DB0
PA7	9	32	DB1
PB0	10	31	DB2
PB1	11	30	DB3
PB2	12	29	DB4
PB3	13	28	DB5
PB4	14	27	DB6
PB5	15	26	DB7
PB6	16	25	Ø2 IN
PB7	17	24	$\overline{FLAG}$
$\overline{PC}$	18	23	$\overline{CS}$
TOD	19	22	R/$\overline{W}$
VCC	20	21	$\overline{IRQ}$

DIE KONTROLLPORTS (Abfrage von Joysticks und Paddles)

Der Commodore 64 verfuegt, im Gegensatz zum VIC-20, ueber zwei Kontrollports, die sich links vom Netzschalter befinden. An jeden dieser Ports laesst sich entweder ein Joystick oder ein Paar Paddles anschliessen. Damit ist es zum Beispiel jetzt auch bei Spielen fuer zwei Personen moeglich, jedem Spieler einen Joystick zuzuordnen. Im folgenden wird dem linken Kontrollport die Nummer 1, dem rechten die Nummer 2 zugeordnet (in Uebereinstimmung mit den Beschriftungen am Geraet selbst). Die beiden Ports sind in Bezug auf die Abfrage von Joystick und Paddles funktionsgleich, Port 1 kann jedoch zusaetzlich zum Anschluss eines Lightpens verwendet werden. Doch laesst sich diese spezielle Eigenschaft des ersten Ports vielleich von Fall zu Fall auch im Zusammenhang mit dem Joystick verwenden (siehe Erlaeuterung).

Der Joystick

Jeder Joystick ist ueber fuenf Signalleitungen, die alle an die IRQ-CIA 6526 (CIA #1) fuehren, mit dem Computer verbunden. Dabei handelt es sich um die vier Leitungen fuer die vier Hauptrichtungen des Joysticks sowie um eine Leitung fuer den Feuerknopf. Die Joystickleitungen fuer den ersten Kontrollport sind an den Datenport B der CIA #1 (U2) angeschlossen, die des zweiten Kontrollports sind mit Port A der gleichen CIA verbunden. Hier die genaue Uebersicht ueber die Bedeutungen der Portleitungen:

	Kontrollport #1	Kontrollport #2
oben:	Port B0	Port A0
unten:	Port B1	Port A1
links:	Port B2	Port A2
rechts:	Port B3	Port A3
Feuer:	Port B4	Port A4

Dass die zugehoerigen Datenrichtungsregister auf Eingang (Bitwert 0) geschaltet wurden, ist die Voraussetzung fuer ein einwandfreies Funktionieren jeder Abfrageroutine von Datenports.

Wird der Joystick nun in eine bestimmte Richtung gehalten (beziehungsweise der Feuerknopf gedrueckt), so ist das entsprechende Bit des Datenports geloescht, ansonsten gesetzt. Bei Diagonalrichtungen sind entsprechend die zwei Richtungsbits, aus denen sich die Diagonalrichtung zusammensetzt, geloescht.

Allerdings hat dieses ganze Prinzip einen recht gewichtigen Haken, der bestimmt jedem schon aufgefallen ist, der (bei eingabebereitem Computer) mit dem Joystick experimentiert hat: Es erscheinen Zeichen auf dem Bildschirm, obwohl keinerlei Taste gedrueckt wird (bei Anschluss an Kontrollport 1) oder einzelne Tasten funktionieren nicht oder erzeugen falsche Zeichen (bei Anschluss an Kontrollport 2), wenn der Joystick benutzt wird. Dies resultiert aus der gemeinsamen Benutzung der Datenleitungen der IRQ-CIA sowohl fuer Joystick als auch Tastatur. Daher ist eine gleichzeitige (parallele) Benutzung von Joystick und Tastatur nicht moeglich.

Das folgende kurze Beispielprogramm fragt den Joystick des ersten Kontrollports ab und gibt entsprechend Meldung auf dem Bildschirm ab:

```
100 DIM A$ (4) : FOR I = 0 TO 4 : READ A$ (I) : NEXT
110 DATA OBEN, UNTEN, LINKS, RECHTS, FEUER
120 B = PEEK (56323) : POKE 56323, B AND 224
130 P = PEEK (56321) : POKE 56323, B
140 IF (P AND 31) = 31 GOTO 120
150 FOR I = 0 TO 4
160 IF (P AND 2 ↑ I) = 0 THEN PRINT A$ (I) " ";
170 NEXT : PRINT : GOTO 120
```

Um den zweiten Joystick abzufragen, muessen lediglich die Adressen 56321 in 56320 sowie 56323 in 56322 geaendert werden. Da die Anordnung der Leitungen identisch ist, ergeben sich keine weiteren Aenderungen.

Die Abfrage der Diagonalrichtungen kann uebrigens gleichzeitig mit der Abfrage der vier Hauptrichtungen erfolgen, sodass nicht alle acht moeglichen Richtungen getrennt erkannt werden muessen: Soll zum Beispiel eine Spielfigur durch den Joystick gelenkt werden, so koennen durch aufeinanderfolgendes Herausfiltern der einzelnen Bits (und, falls ein Bit geloescht ist, Verschiebung in die entsprechende Richtung) die Diagonalrichtungen gleich mitberuecksichtigt werden.

Hierzu ein kurzes Beispiel:

Die Position einer Figur (zum Beispiel eines Sprites) wird durch die Variablen X und Y festgelegt. Abfrage des Joysticks an Port 1 und Aenderung der entsprechenden Variablen durch folgendes Programm:

```
B = PEEK (56323) : POKE 56323, B AND 224
P = PEEK (56321) : POKE 56323, B
IF (P AND 1) = 0 THEN Y = Y - 1
IF (P AND 2) = 0 THEN Y = Y + 1
IF (P AND 4) = 0 THEN X = X - 1
IF (P AND 8) = 0 THEN X = X + 1
```

In Maschinensprache laesst sich dies durch das Herausshiften von Bits und Abfrage durch die Carry (eventuell innerhalb einer Schleife mit vier Durchlaeufen) und indiziertem Laden des Wertes fuer die Erhoehung (Verminderung) des Wertes fuer die Position natuerlich wesentlich eleganter und kuerzer (und natuerlich schneller) erreichen. Die Abfrage der Feuertaste hat sinnvollerweise getrennt zu erfolgen.

Sollten sich aus der Tastaturabfrage des Betriebssystems durch die Routine SCNKEY im Zusammenhang mit der Abfrage der Joysticks Probleme ergeben (die eigene Routine wird beeinflusst oder aehnliches), so kann die Interruptroutine (generell) abgeschaltet werden durch ...

```
POKE 56333, 1
```

Wieder eingeschaltet wird der IRQ durch ...

```
POKE 56333, 129
```

Allerdings kann waehrend dieser Zeit, in der IRQs nicht zugelassen sind, keinerlei Abfrage der Tastatur durch GET oder INPUT erfolgen (aber dies ist durch die gleichzeitige Benutzung des Joysticks wohl gar nicht erwuenscht beziehungsweise moeglich). Auch sollte nach Beendigung der Abfrage des Joysticks der IRQ unbedingt wieder zugelassen werden. Ist das Programm naemlich beendet, so koennen ansonsten keinerlei Eingaben gemacht werden, da keine Tastendruecke mehr erkannt werden.

Waehrend der Zeit abgeschalteten IRQs laeuft die interne (softwaremaessig realisierte) Uhr NICHT weiter. Auch die RUNSTOP-Taste wird nicht mehr abgefragt. RUNSTOP im Zusammenhang mit RESTORE ist jedoch auch weiterhin moeglich.

Die Paddles

An einen Kontrollport laesst sich je ein Paar Paddles anschliessen. Da jedoch der Sound-Baustein (SID 6581) des Commodore 64 nur ueber zwei Analog/Digital-Wandler verfuegt, muss vor der Abfrage der Paddles angegeben werden, welches Paddlepaar bei der Abfrage gemeint ist. Nur die Feuertasten sind gleichzeitig abfragbar, da diese schliesslich nicht von einem A/D-Wandler abhaengig sind, sondern (wie auch die Joysticks) direkt mit den Ports der IRQ-CIA verbunden sind. Diese beiden 8-Bit-Register (POT X und POT Y genannt) des SID 6581 fuer die Paddles geben die Stellung der Drehregler an.

Die beiden Paddles werden gemaess der Position ihres Verbindungskabels am Stecker fuer den Kontrollport mit 'Paddle links' und 'Paddle rechts' bezeichnet. Es werden folgende Register verwendet:

	Paddle links:	Paddle rechts:
Drehregler:	SID 6581, POT Y	SID 6581, POT X
Feuerknopf:	CIA 6526, Portbit 2	CIA 6526, Portbit 3

Auch hier gilt, dass das Bit fuer den Feuerknopf geloescht ist, wenn die Taste gedrueckt ist.

Wie erkennbar, werden die gleichen Leitungen benutzt, wie von den Joysticks auch. Da aber an einen Kontrollport nicht gleichzeitig Joystick und Paddle anschliessbar sind, resultiert daraus auch kein Nachteil. Allerdings gelten auch hier die in Bezug auf Tastatur und Joystick gemachten Einschraenkungen durch die gemeinsame Benutzung von Datenleitungen.

Zuerst zu den Feuerknoepfen: Wie dies auch bei den Joysticks der Fall war, so ist auch hier dem linken Paddlepaar der Port B und dem rechten der Port A zugeordnet. Mit 'Portbit' in der obigen Uebersicht ist also jeweils das entsprechende Bit im zugeordneten Datenport (A oder B) gemeint. Hier noch einmal die Adressen der Register der CIA fuer die Kontrollports:

56320: Portregister A
56321: Portregister B
56322: Datenrichtungsregister A
56323: Datenrichtungsregister B

Auch hier muessen natuerlich die entsprechenden Bits des Datenrichtungsregisters auf Eingang geschaltet werden, um die Feuerknoepfe abzufragen.

Befindet sich ein Drehregler am RECHTEN Anschlag, so enthaelt das entsprechende Register (SID 6581) den Wert 0, am LINKEN Anschlag einen Wert um 200 (Commodore-Paddles), in allen anderen Positionen einen entsprechenden Zwischenwert (diese Werte unterliegen auch bei konstanter Reglerstellung gewissen Schwankungen). Es ist moeglich, dass sich die Reglerwerte von Paddle zu Paddle etwas unterscheiden, dies sollte jedoch in Programmen zu keinerlei Problemen fuehren. Eventuell kann in einem Programm aber auch festgestellt werden (durch Anweisung an den Benutzer, den Regler zum linken

Der erste Wert wird nun in die Adresse 43, der zweite in die Adresse 44 geschlrieben. Nun kann der LOAD-Befehl (ohne Angabe einer Sekundaeradresse) vorgenommen werden (das zu ladende Programm muss natuerlich ein Relativprogramm sein). Ist der Ladevorgang abgeschlossen, so muss der Pointer (43/44) wieder auf den Anfang des ersten Programms gesetzt werden. Dies erreicht man durch ...

```
POKE 43, 1 : POKE 44, 8
```

Sie haben so erwirkt, dass beide Programme zu einem Programm zusammengefuegt wurden.

Der Dateityp 5, der End-of-Tape-Block, wird zur Bandende-Kennzeichnung verwendet und wird oft hinter dem letzten File einer Cassettenseite abgespeichert, um unnoetiges weiteres Suchen zu vermeiden. Wird beim Laden anstelle eines Daten- oder Programmheaders ein solcher EOT-Block gefunden, so sollte eigentlich "?FILE NOT FOUND ERROR" ausgegeben werden. Allerdings geschieht dies aufgrund eines Fehlers im Betriebssystem, der auch schon beim VIC-20 aufgetreten ist, nicht. Es wird die in diesem Zusammenhang sinnlose Meldung "?DEVICE NOT PRESENT ERROR" gegeben (vgl. OS-Listing an den Stellen 62383 ff, 62806 ff, 63291 f sowie 57717 f).

Wie koennen nun diese ganzen verschiedenen Dateiarten (und Bloecke) auf Band geschrieben werden? Hier eine Uebersicht darueber:

SAVE "PROGRAMMNAME"	Abspeicherung eines Programms als Relativprogramm (meist BASIC)
SAVE "PROGRAMMNAME",1,1	wie oben, jedoch wird zusaetzlich ein EOT-Block am Ende gespeichert
SAVE "PROGRAMMNAME",1,2	Abspeicherung eines Programms als Absolutprogramm (meist 6502-Code)
SAVE "PROGRAMMNAME",1,3	wie oben, jedoch wird zusaetzlich ein EOT-Block am Ende gespeichert

Bei den letzten beiden Beispielen wird es meist noch noetig sein (falls das Maschinenprogramm auf diese Weise und nicht ueber eine Maschinenroutine gespeichert wird), die Start- und Endadresse (plus eins) des Maschinenprogramms festzulegen. Dies erreicht man durch Zerlegung beider Adressen jeweils in LSB und MSB. Diese muessen dann in den Pointer fuer die Startadresse (43/44) und Endadresse (45/46) von BASIC-Programmen geschrieben werden. Dann kann das SAVE-Kommando gegeben werden.

Beim Eroeffnen von Files auf Cassette gibt es folgende drei Moeglichkeiten:

OPEN 1, 1, 0, "FILENAME"	Eroeffnung eines Lesefiles
OPEN 1, 1, 1, "FILENAME"	Eroeffnung eines Schreibfiles
OPEN 1, 1, 2, "FILENAME"	Eroeffnung eines Schreibfiles mit zusaetzlicher Abspeicherung eines EOT-Blocks nach "CLOSE"

Beim Laden von Programmen, dazu gehoert auch der VERIFY-Befehl, gibt es zwei Arten des Ladens:

Lightpen (Lightgun)

An den ersten Kontrollport kann ein Lightpen angeschlossen werden. Dieser setzt, wenn seine Elektronik den Zeilenstrahl, der das Bild auf dem Bildschirm aufbaut, erkennt, die Leitung fuer die Feuertaste des Joysticks auf Lowpegel. Da die Steuerleitung fuer die Feuertaste jedoch nicht nur mit der IRQ-CIA verbunden ist, sondern auch mit dem Lightpen-Input (active low), wird die Position des Zeilenstrahls im Augenblick der negativen Flanke vom VIC-II-Chip aus in ein anderes Register uebertragen, das dann zur Erkennung der Position des Lightpens dient. Ausserdem wird bei aktivem Lightpen-Input ein Interrupt ausgeloest (sofern zugelassen), sodass das Computerprogramm nur dann die Lightpen-Position feststellen muss, wenn es ueberhaupt noetig ist (per Interruptprogramm). Eine Lightgun wird jedoch noch mehr Verbindungen zum Computer besitzen. So zum Beispiel eine Feuertaste, die dann auch wirklich diese Funktion hat (dann aber natuerlich nicht mehr mit der normalen Feuerleitung abgefragt werden kann), da die Leitung zur Feststellung der Position wohl fuer ein Spiel nicht ausreicht.

Da der Joystick des ersten Kontrollports jedoch auch mit diesem Lightpen-Input verbunden ist, laesst sich dadurch die Feuertaste (zum Beispiel waehrend des Demo-Modus' bei einem Spielprogramm oder nach Beendigung des Spiels) auch zum Neustart des Spiels verwenden, ohne dass die Taste abgefragt werden muss. Es wird nur eine Interruptroutine benoetigt (die beim Druecken der Feuertaste aufgerufen wird, wenn das zugehoerige Interrupt-Enable-Bit gesetzt wurde), die dann einfach einen Neustart vornimmt. Natuerlich ist diese Anwendung nur dann sinnvoll, wenn eine gewoehnliche Abfrage der Feuertaste aufgrund der zeitlichen Ausdehnung einiger Programmroutinen den Benutzer zu lange warten liesse. Man koennte zwar in jeder Routine die Taste abfragen, sodass niemals zu lange auf eine Reaktion gewartet werden muesste, dies koennte jedoch den Programmablauf verlangsamen oder zu speicherplatzaufwendig sein.

Detailliertere Informationen in Bezug auf die Auswirkungen und Zusammenhaenge des Lightpen-Inputs auf den VIC-II-Chip finden sich bei der Erklaerung des Video-Chips.

DATENSPEICHERUNG AUF CASSETTE UND DISK

Der Commodore 64 bietet recht komfortable Befehle zur Datenspeicherung, wozu auch die Speicherung von Programmen gehoert. Dabei stehen Ihnen prizipiell zwei vom Betriebssystem unterstuetzte Moeglichkeiten zur Verfuegung:

1. die Speicherung auf Cassette sowie

2. die Speicherung auf einem Geraet, das an den seriellen Bus anschliessbar ist, meist wohl ein Diskettenlaufwerk.

Der Cassettenrecorder ist das wohl guenstigste Medium, um Daten (meist Programme) zu speichern. Dies bringt jedoch einige erhebliche Nachteile mit sich. Der wohl erheblichste Nachteil ist, dass Daten nur direkt hintereinander geschrieben und in der gleichen Reihenfolge auch wieder gelesen werden koennen. Genausowenig koennen Daten an eine Datei angefuegt oder Inhalte einer Datei geaendert werden. Dazu muesste die gesamte Datei gelesen, modifiziert und wieder auf Band geschrieben werden. Waehrend dieser Nachteil bei Programmen nicht so gravierend ist (schliesslich hat man ja ein Zaehlwerk, an dem man sich orientieren kann), so ist dies doch bei der Datenspeicherung, speziell bei voneinander abhaengigen Dateien, ein solches Handicap, sodass bei komplexeren Dateiverwaltungen auf eine Diskettenstation wohl nicht verzichtet werden kann.

Ein weiterer, schwerwiegender Nachteil ist die Geschwindigkeit der Datenuebertragung auf Band. Waehrend selbst ein langes Programm in nur einigen Sekunden von Diskette gelesen werden kann, so koennen beim Laden von Cassette schon einige Minuten (zuzueglich der Zeit fuer das Umspulen der Cassette) zusammenkommen.

Datenspeicherung auf Cassette

Sollen Daten auf Band geschrieben werden, so wird erst einmal ein Vorspann, bestehend aus einem hohen Ton, auf Cassette geschrieben. Darauf folgt ein sogenannter "Header", der (genauso wie das auf den Header folgende Programm oder der Datenblock) zweimal hintereinander geschrieben wird, um Fehler, die beim Lesen eventuell auftreten, korrigieren zu koennen. Die Commodore-Computer koennen maximal 31 Fehler pro Block (das kann ein Header, ein ganzes Programm oder ein Datenblock sein) korrigieren, wenn im zweiten Block diese Daten lesbar vorhanden sind. Werden auch beim Lesen des zweiten Blocks Fehler in den Bytes, die im ersten Block nicht lesbar waren, erkannt, so wird Bit 4 der Statusvariablen ST gesetzt. Ist ein Blockende erkannt worden, ohne dass alle Daten gelesen werden konnten, so wird Bit 2 (short Block) gesetzt. Entsprechend wird Bit 3 (long Block) gesetzt, wenn bereits alle Daten gelesen wurden, ohne dass ein Blockende erkannt wurde. Als eine weitere Fehlermoeglichkeit besteht der sogenannte Pruefsummenfehler (Bit 5). Dieser tritt auf, wenn die beim Abspeichern des Programms mitgegebene Pruefsumme (sie entsteht aus der Exklusiv-Oder-Verknuepfung saemtlicher Datenbytes) nicht mit der beim Lesen errechneten (erwarteten) Pruefsumme uebereinstimmt.

Bei der Speicherung von Daten werden diese vor der Abspeicherung erst einmal in einem Puffer von 192 Bytes im Bereich von 828 bis 1019 gesammelt, der dann, wenn er gefuellt ist (beziehungsweise, wenn das File geschlossen wird), auf Band geschrieben wird. Aehnliches erfolgt beim Einlesen: Es wird immer ein ganzer Datenblock auf einmal gelesen (auch wieder

in den Bereich von 828 bis 1019), aus dem dann die Daten stueckweise durch Befehle wie GET# und INPUT# geholt werden. Wuerden die Daten immer direkt auf Band geschrieben (beziehungsweise von Band geholt werden), so muesste man jedesmal den Vorspann, die Synchronisationsbytes und alles andere mitschreiben (da sonst die Daten nicht fehlerfrei gelesen werden koennten, schliesslich muss der Recordermotor erst wieder anlaufen), egal wie kurz der Block waere. Da dadurch jedoch die Geschwindigkeit noch weiter herabgesetzt werden wuerde, hat man sich wohl fuer das Puffer-Prinzip entschieden.

Der eben schon erwaehnte Header enthaelt die Informationen darueber, um welche Art von Datenfile es sich bei dem folgenden handelt. Ausserdem ist in ihm der Programmname (oder Filename) und, falls es sich um ein Programm handelt, die Start- und Endadresse enthalten. Der Header wird, genauso wie die Datensaetze (die prizipiell ihr eigener Header sind), im Cassettenpuffer aufgebaut und dorthin auch wieder eingelesen.

Hier der Aufbau des Headers:

```
Adresse          828: Kennzeichnung fuer Dateityp
        829 bis  830: Startadresse des Programms (low, high)
        831 bis  832: Endadresse+1 des Programms (low, high)
        833 bis  848: Filename (Ausgabe bei "FOUND")
        849 bis 1019: Filename (keinerlei Ausgabe)
```

Wird beim Abspeichern eines Programms (oder beim schreibenden Eroeffnen eines Files) ein Filename (bestehend aus mehr als 16 Zeichen) angegeben, so werden diese "ueberschuessigen" Zeichen durchaus abgespeichert, da maximal 187 Zeichen als Filename zugelassen werden. Allerdings werden beim Einlesen nur noch die ersten 16 Zeichen ausgegeben, die restlichen bleiben unsichtbar, koennen jedoch mittels CHR$ und PEEK ausgelesen werden. Diese weiteren Zeichen werden jedoch beachtet, wenn beim lesenden Zugriff auf Band, sei es durch LOAD, VERIFY oder OPEN, der Filename des zu suchenden Files angegeben wird. Ist dieser naemlich laenger als 16 Zeichen, so wird fuer den Vergleich, ob das richtige File gefunden wurde, auf den "unsichtbaren" Rest des Filenamens zugegriffen. Die Verwendung von Filenamen mit mehr als 16 Zeichen ist jedoch bei der normalen Verwendung des Bandgeraets unueblich.

Als Kennzeichnung fuer den Dateityp (Adresse 828) stehen folgende Moeglichkeiten zur Auswahl:

1: Relativprogramm (wird an BASIC-Startadresse geladen)
2: Datenblock, folgende 191 Bytes sind Daten (Filehandling)
3: Absolutprogramm, wird an angegebene Stelle geladen
4: Dateiheader, folgende Bloecke sind Datenbloecke
5: End of Tape (EOT) Block, Kennzeichnung fuer Bandende

Der Dateityp 2 gibt an, dass es sich bei diesem Block um Daten handelt, die mittels PRINT# auf Band geschrieben wurden und durch GET# und INPUT# gelesen werden koennen.

Dieser Blocktyp enthaelt selbst keinen Programmnamen, dieser steht zu Beginn des Files (es koennen mehrere Bloecke vom Typ 2 hintereinander stehen, je nach Laenge der Datei) im Dateiheader (Headertyp 4), der beim lesenden Zugriff unbedingt gefunden werden muss, da der Computer ansonsten nichts mit den folgenden Datenbloecken anfangen kann.

Die Dateitypen 1 und 3 werden fuer die Speicherung von Programmen verwendet. In diesen beiden Faellen sind auch die vier Bytes nach dem Kennungsbyte innerhalb des Dateiheaders benutzt. Sie enthalten die Startadresse und die Endadresse (plus eins, wie dies bei den meisten Endpointern der Fall ist) des Programms. Diese wird beim Schreiben auf Band festgestellt und dem Header mitgegeben.

Handelt es sich bei dem gelesenen Header um Typ 3, so wird das Programm an die Adresse geladen, an der es sich befand, als es auf Cassette gespeichert wurde. Dies kann bei Maschinenprogrammen sehr nuetzlich sein, da dann das Transferieren in den Speicherbereich, in dem sie ablaufen sollen, entfaellt. Auch koennen solcherart abgespeicherte Programme nachtraeglich eingeladen werden, ohne dass ein schon im Arbeitsspeicher befindliches BASIC-Programm (sofern es nicht in den vom Maschinenprogramm belegten Bereich hineinreicht) ueberschrieben wird. Allerdings muss auch hier die folgende Vorgehensweise beachtet werden:

Vor dem Einladen des Programms muss der Wert des Pointers (45/46) gemerkt werden. Dies kann einfach durch PEEK erfolgen. Nun koennen Sie das gewuenschte Maschinenprogramm, das, wie gesagt, nicht im gleichen Bereich liegen darf wie das schon im Arbeitsspeicher befindliche BASIC-Programm, nachladen. Ist der Ladevorgang abgeschlossen, so stellen sie den alten Wert des Pointers (45/46) durch POKE wieder her. Diesen POKEs muss unbedingt (!) ein CLR folgen, damit auch die zugehoerigen anderen Pointer richtig gesetzt werden. Auch muss bedacht werden, dass das eingeladene Maschinenprogramm (je nach Lage) vielleicht vor dem Zugriff durch das Betriebssystem geschuetzt werden muss.

Bei dem Dateityp 1 handelt es sich meist um BASIC-Programme, deren Startadresse (im Normalfall) unbeachtet bleibt. Stattdessen werden sie an die Startadresse des BASIC-Bereichs (Standardwert 2049) geladen. Allerdings kann auch bei diesem Programmtyp ein Laden in den urspruenglichen Programmbereich erzwungen werden. Dies erreicht man durch einen LOAD-Befehl mit Sekundaeradresse ungleich null.

Durch den Programmtyp des Relativprogramms kann ein Zusammenhaengen von Programmen erreicht werden. Dazu muss einfach der Pointer auf die Startadresse von BASIC-Programmen (Adressen 43 und 44) auf das Ende des im Arbeitsspeicher befindlichen Programms gesetzt werden. Allerdings muss das anzuhaengende Programm groessere Zeilennummern haben, als das im Arbeitsspeicher befindliche Programm. Hier die genaue Angabe der Methode:

Feststellung der Endadresse des Programms im Arbeitsspeicher durch ...

```
E = PEEK (45) + 256 * PEEK (46)
```

Von diesem Wert muss nun der Wert zwei subtrahiert werden (siehe: Interne Codierung von BASIC-Programmen) ...

```
E = E - 2
```

Dieser Wert muss nun wieder in zwei Bytes zerlegt werden. Dies geschieht durch ...

```
PRINT E - 256 * INT (E / 256), INT (E / 256)
```

und rechten Anschlag zu drehen), wo die Minimal- und Maximalwerte liegen.

Die Register, aus denen die Werte fuer die Drehregler ausgelesen werden koennen, befinden sich an den Adressen 54297 (POT X, Paddle rechts) und 54298 (POT Y, Paddle links). Da wohl nur selten alle 256 Werte (oder 200, siehe oben) sich in der Darstellung auf dem Bildschirm voneinander unterscheiden werden, ist es wohl sinnvoll, entweder nur Werte innerhalb bestimmter Grenzen zu verarbeiten und darueber hinausgehende Werte als Maximal- beziehungsweise Minimalwert zu betrachten. Eine andere - jedoch aufwendigere - Methode waere, den Wert aus dem Register so zu dividieren, dass genau die Spanne an Werten herauskommt, die fuer die Verarbeitung sinnvoll ist.

Jetzt fehlt nur die Information darueber, wie die beiden Paddlepaare bei Abfrage des Reglerwertes ausgewaehlt werden koennen. Die Auswahl erfolgt ueber Bit 6 und 7 von Port A der IRQ-CIA. Wie sofort erkennbar, entstehen auch hier wieder Konflikte mit der Tastatur, die sich hier jedoch durch Abschalten des Interrupts verhindern lassen.

Ist Bit 6 dieses Ports gesetzt und Bit 7 geloescht, so sind die Register POT X und POT Y des SID 6581 dem linken Paddlepaar (Kontrollport 1) zugeordnet. Ist dahingegen Bit 6 geloescht und Bit 7 gesetzt, so kann die Reglerstellung des rechten Paddlepaares (Kontrollport 2) festgestellt werden. Nach so viel theoretischer Erklaerung auch hierzu wieder ein Beispielprogramm ...

```
100 PA = 56320 : PB = 56321 : DB = 56323 : CA = 56333
110 PY = 54298 : PRINT CHR$ (147)
120 PRINT " KONTROLLPORT 1   KONTROLLPORT 2
130 PRINT "     PADDLE           PADDLE
140 PRINT " LINKS   RECHTS   LINKS   RECHTS" : PRINT : PRINT
150 PRINT CHR$ (145); : POKE CA, 1 : FOR I = 0 TO 1
160 POKE PA, 127 + 64 * I : D = PEEK (DB - I)
170 POKE DB - I, D AND 243 : P = PEEK (PB - I)
180 POKE DB - I, D : PRINT " "; : FOR J = 0 TO 1
190 IF (P AND 2 ↑ (3 - J)) = 0 THEN PRINT CHR$ (18);
200 PRINT RIGHT$ ("   " + STR$ (PEEK (PY - J)), 4);
210 PRINT " " CHR$ (146) SPC(3); : NEXT : NEXT : PRINT
220 POKE CA, 129 : GET G$ : IF G$ <> CHR$ (13) GOTO 150
```

Das Programm gibt zu jedem der vier moeglichen Paddles den jeweiligen Wert des Drehreglers an. Ist ausserdem die Feuertaste gedrueckt, so wird der Wert invertiert (in Negativschrift) dargestellt. Das Programm laesst sich durch Druekken von RETURN abbrechen (durch den abgeschalteten IRQ muss die Taste eventuell laenger gedrueckt gehalten werden).

LOAD "PROGRAMMNAME" Lesen eines Programms, wie es im Programmheader angegeben wurde. (also Absolut oder Relativ)

LOAD "PROGRAMMNAME",1,1 Absolutes Einladen eines Programms in den im Header spezifizierten Speicherbereich

Noch eine Anmerkung zur Speicherung (speziell von Maschinenprogrammen): Der Commdore 64 kann, im Gegensatz zu allen vorherigen Modellen von Commodore, ohne zusaetzliche Hilfsprogramme auch den Speicherbereich ab 32768 auf Cassette speichern. Dies ist beim Commodore 64 noetig geworden, da BASIC-Programme ja auch in den Bereich ueber 32767 hineinreichen koennen. So ist es aber nun zum Beispiel auch moeglich, den 4-KB-RAM-Bereich von 49152 bis 53247, in dem sich viele Maschinenutilities unterbringen lassen, auf Cassette zu speichern.

Datenspeicherung auf Diskette

Der Vorteil beim Arbeiten mit Floppy-Disc liegt darin, dass beliebig auf jeden einzelnen Block einer Diskette zugegriffen werden kann. So muss auch beim Lesen und Schreiben auf Diskette nicht erst der Kopf durch getrennte Befehle positioniert werden. Dies alles erfolgt automatisch durch das Diskettenbetriebssystem. Wird ein verlangtes File nicht gefunden, so wird die Fehlermeldung "?FILE NOT FOUND ERROR" gegeben.

Beim Abspeichern von Programmen auf Diskette kann nicht spezifiziert werden, ob es sich um ein Maschinenprogramm handelt (die Angabe eines EOT-Blocks ist sowieso unsinnig). Dies wird beim Einladen angegeben, aehnlich, wie es auch bei Cassette moeglich war:

LOAD "PROGRAMMNAME",8 Einladen eines Programms an den Speicheranfang (BASIC-Programme)

LOAD "PROGRAMMNAME",8,1 Einladen eines Programms an den beim Abspeichern angegebenen Speicherbereich

Auch beim Laden von Diskette ist das Zusammenhaengen von BASIC-Programmen moeglich. Hierzu muss genauso vorgegangen werden, wie beim APPEND von Cassette. Die Abspeicherung von Maschinenprogrammen erfolgt wie bei Cassette auch, jedoch wird bei der Abspeicherung auf Diskette keinerlei Sekundaeradresse angegeben. Dies ist zwar moeglich, es hat aber keinerlei Auswirkung auf den Abspeicherungsvorgang.

Nun zu den internen Vorgaengen beim Abspeichern und Einlesen von Programmen im Zusammenhang mit Diskette:

Wird ein Programm auf Diskette geschrieben, so wird ein File mit der Sekundaeradresse 1 eroeffnet. Dies ist fuer die Commodore-Laufwerke der Hinweis dafuer, dass ein Datenfile vom Typ "PRG" eroeffnet werden soll. Die ersten beiden Bytes, die dann auf dieses File ausgegeben werden, sind die Startadresse des Programms in Low/High-Darstellung. Eine Endadresse wird nicht mitgegeben. Das Programmende bei der Datenuebertragung wird der Floppy-Disk durch ein EOI mitgeteilt.

Das Einlesen erfolgt durch Eroeffnung eines Files mit der Sekundaeradlresse 0. Auch diese spezielle Sekundaeradresse

ist ein Hinweis fuer die Diskettenstation: Es wird ein File vom Typ "PRG" zum Lesen eroeffnet. Dann werden die ersten beiden Bytes gelesen und als Ladezeiger gespeichert. Diese beiden Bytes werden jedoch nur dann auch als Ladezeiger verwendet, wenn eine Sekundaeradresse ungleich null dem LOAD-Kommando mitgegeben wurde. Ansonsten wird das Programm in den fuer BASIC-Programme vorbereiteten Bereich geladen.

So kann zum Beispiel die Startadresse eines Programms auf Diskette durch folgendes kurzes Segment festgestellt werden:

```
OPEN 1, 8, 0, "PROGRAMMNAME" : GET#1, L$, H$ : CLOSE 1
PRINT ASC (L$ + CHR$ (0)) + 256 * ASC (H$ + CHR$(0))
```

Durch Angabe der Sekundaeradresse 0 wird der LOAD-Befehl simuliert, wodurch weitere Angaben innerhalb des Filenamens entfallen. Der Zusatz 'CHR$(0)' wird benoetigt, da das Betriebssystem bei GET# den Leerstring anstelle des Nullcodes uebermittelt. Der ASC ist fuer den Leerstring allerdings nicht definiert. Daher wird diese Stringaddition vorgenommen.

OVERLAY (Nachladen von Programmen unter Programmkontrolle)

Wird der LOAD-Befehl (unabhaengig vom angesprochenen Geraet) innerhalb eines Programms gegeben, so wird nach dem Einladen des Programms der CHRGET-Pointer auf den Anfang des BASIC-Bereichs gesetzt. Ein nachgeladenes Programm wird daher sofort gestartet. Aber Achtung: Um die Variableninhalte nicht zu zerstoeren (sie koennten im nachgeladenen Programm ja noch benoetigt werden), wird keinerlei CLR ausgefuehrt. Auch wird der BASIC-Endepointer (Variablenanfangspointer) nicht gesetzt, er behaelt seinen alten Wert bei. Ein nachgeladenes Programm darf daher niemals laenger als das aufrufende Programm sein. Auch muss beachtet werden, dass ein nachgeladenes Programm niemals wieder abgespeichert werden sollte. Schliesslich stimmen die Endezeiger nicht, so dass das Programm mehr Platz belegen wuerde, als wirklich notwendig waere.

Hier noch die Einschraenkungen bei der Uebernahme von Variablen: Stringinhalte, die direkt zugewiesen wurden, werden nicht uebertragen (der Stringdescriptor zeigt direkt in den BASIC-Text). Dies kann jedoch durch Addition des Leerstrings umgangen werden. Also beispielsweise ...

```
A$ = "STRINGINHALT" + ""
```

... anstelle von ...

```
A$ = "STRINGINHALT"
```

Auch muessen ueber 'DEF' definierte Funktionen neu definiert werden, da auch der Pointer des 'FN-Variableneintrags' direkt in den BASIC-Text zeigt, der jedoch nach dem Nachladen nicht mehr in dieser Form vorhanden ist. Eine genaue Erklaerung der Abspeicherung von FN-Variableneintraegen findet sich bei der Erklaerung der Darstellung und Ablage von Variablen.

Werden die Variableninhalte aus dem aufrufenden Programm jedoch nicht mehr weiter benoetigt und koennen daher geloescht werden, so empfiehlt es sich, das Kommando zum Nachladen durch den Tastaturpuffer zu geben. Dies kann fuer Diskette erfolgen durch ...

```
A$ = "PROGRAMMNAME"
PRINT CHR$ (147) : PRINT
PRINT "LOAD" CHR$ (34) A$ CHR$ (34) ",8"
PRINT : PRINT : PRINT : PRINT : PRINT "RUN" CHR$ (19);
POKE 631, 13 : POKE 632, 13 : POKE 198, 2
END
```

Dieses Programm schreibt den auszufuehrenden Text auf den Bildschirm und laesst diesen dann durch ein simuliertes Druecken der RETURN-Taste interpretieren.

Es ist auch moeglich, wenn der Ladebefehl im Programm gegeben wurde (und nicht ueber den Tastaturpuffer), die BASIC-Endepointer nach dem Pointer (174/175) zu setzen, da dieser die Endadresse des soeben geladenen Programms angibt. Allerdings verlangt diese Programmierung eine Kontrolle, da, wenn das Programm nach einer Aenderung direkt mittels "RUN" gestartet wird, die Endepointer dann falsch gesetzt werden. Eine Moeglichkeit des Abfangens dieses Effektes bestuende darin zu pruefen, ob der Variablenendepointer gleich dem Variablenanfangspointer ist. Dies ist schliesslich nach dem Starten eines Programms durch "RUN" der Fall, waehrend es beim Nachladen nicht der Fall ist (die Variableninhalte bleiben erhalten).

Soll vom Hauptprogramm aus ein Maschinenprogramm nachgeladen werden, so hat das Nichtsetzen der Endepointer den grossen Vorteil, dass das Programm ungestoert weiterlaufen kann. Hier ein Beispiel fuer einen Programmanfang, der ein benoetigtes Maschinenprogramm nachlaedt:

```
100 IF A = 0 THEN A = 1 : LOAD "MASCHINENPROGRAMM", 8
110 ...
```

Da nach dem Starten eines Programms durch "RUN" saemtliche Zahlenvariablen den Wert null haben, werden die Befehle nach der IF-Abfrage ausgefuehrt. Die Variable A wird nun aber auf den Wert eins gesetzt. Nachdem das Maschinenprogramm nachgeladen wurde, werden die CHRGET-Pointer auf den Programmanfang gesetzt, die Variableninhalte jedoch nicht geloescht. Nun hat die Variable A jedoch nicht mehr den Wert null, es wird in der naechsten Zeile mit der Programmausfuehrung fortgefahren.

Dieser Effekt laesst sich auch verwenden, wenn man nachtraeglich zum Beispiel Maschinenutilities nachladen will. Man gibt dazu in das momentan vorhandene BASIC-Programm (das die Zeilen 0 und 1 nicht enthalten darf) nur ein ...

```
0 END
1 LOAD "MASCHINENPROGRAMM"
```

... und startet es durch ...

```
RUN 1
```

Das Maschinenprogramm wird nun nachgeladen, ohne dass etwas weiteres geschieht. Man erspart sich so das Merken und Wiederherstellen der Pointer auf das Ende des BASIC-Programms. Allerdings sollte man die beiden Programmzeilen nach Ausfuehrung wieder entfernen.

PRAKTISCHE ANWENDUNG DER "TIME OF DAY"

Wer die softwaremaessig realisierte Uhr des Commodore 64 (ueber die Systemvariablen TI und TI$ abfragbar) schon einmal ueber laengere Zeit hat arbeiten lassen, der weiss, dass diese nicht unbedingt als genau zu bezeichnen ist (circa zwei Prozent Abweichung), was aber bei dieser Art der Verwirklichung leider nicht vermeidbar ist.

Daher wird man vielleicht eine der in den CIAs eingebauten Uhren einmal verwendet haben. Da diese von der Netzfrequenz getaktet werden, weisen sie eine sehr hohe Ganggenauigkeit auf. Allerdings ist die Benutzung dieser Uhren (die vom hardwaremaessigen Konzept her sehr gut durchdacht sind) mit einigen Problemen verbunden. Das Arbeiten mit den beiden Uhren wird in keiner Weise direkt vom Betriebssystem unterstuetzt. Man muss also eine eigene Abfrageroutine erstellen, die die Inhalte der vier Register in irgendeiner Form so verarbeitet, dass man die Uhrzeit in einem brauchbaren Format vorliegen hat. Auch das Setzen der Uhrzeit kann nicht mehr einfach durch Zuweisung erfolgen. Zuerst muss die Uhrzeit in einzelne Ziffern zerlegt werden, die dann abgespeichert werden. Abgesehen davon werden Stunden nicht mehr im Bereich von 0 Uhr bis 23 Uhr, sondern von 1 Uhr bis 12 Uhr angegeben (zusammen mit dem Hinweis, ob es Vormittag oder Nachmittag ist). Man wird daher, so gut das BASIC-Unterprogramm auch sein man, nicht mit dem Komfort arbeiten koennen, wie man ihn von "TI$" her gewohnt ist. Ausserdem tritt (jedenfalls bei einer Netzfrequenz von 50 Hertz) ein weiteres Problem auf: Sobald man ein Programm mittels "RUNSTOP/RESTORE" abbricht, geht die Uhr auffaellig nach. Dies liegt daran, dass die Flag fuer die Taktfrequenz (MSB von CRA) beim Druecken von "RUNSTOP/RESTORE" sofort auf 60 Hertz zurueckgesetzt wird. Man muesste also bei jedem Druecken dieser beiden Tasten diese Flag wieder neu setzen.

Ausserdem scheint bei der Entwicklung der CIAs ein Fehler (?) aufgetreten zu sein: Versucht man das Stundenregister auf 12 Uhr zu setzen, so wird der gespeicherte Flagwert fuer AM/PM sofort umgekehrt. Stellt man also 12 Uhr nachmittags ein, so zeigt die Uhr zwoelf Uhr vormittags (nachts) an (und umgekehrt).

All dieses wurde durch ein kurzes Maschinenprogramm beseitigt: Man kann genauso komfortabel (wenn nicht komfortabler) mit der eingebauten Uhr arbeiten, wie mit "TI$" auch. Es ist moeglich, die Uhrzeit festzusetzen und sie wieder auszulesen. All dies geschieht ueber die USR-Funktion des Commodore-BASICs. Um die Uhrzeit festzusetzen, muss einfach nur eingegeben werden ...

A = USR ("hhmmsst")

Dabei steht "hh" fuer die Stunden (von 0 bis 23!), "mm" fuer die Minuten, "ss" fuer die Sekunden (beides von 0 bis 59) sowie "t" fuer die Zehntelsekunden. Es existiert also eine Stelle mehr, als bei "TI$". Ist die Uhrzeit korrekt angegeben worden, so enthaelt die Variable "A" den Wert null. Wurde das Format missachtet (falsche Laenge, keine Ziffern, falscher Bereich fuer Stunden, Minuten oder Sekunden), so enthaelt "A" den Wert "-1". So ist also keine Ueberpruefung der Eingabe durch das Programm noetig. Es wird der Wert mittels USR an das Maschinenprogramm uebergeben, worauf dieses dann meldet, ob die Eingabe korrekt war. Fuer "A" kann natuerlich jede andere numerische Variable stehen.

Das Lesen der Uhrzeit erfolgt durch ...

A$ = USR (beliebiger numerischer Ausdruck)

Anstelle von A$ kann natuerlich eine andere Stringvariable stehen. Auch die Weiterverarbeitung des Ergebnisses innerhalb eines Ausdrucks ist moeglich. Die Funktion liefert die Uhrzeit in genau dem Format, wie es auch eingegeben werden muss.

Auch das Druecken von "RESTORE" ist mit keinerlei Gefahren fuer die Genauigkeit mehr verbunden, da das Programm ausser dem USR-Vektor auch noch den NMI-Vektor "umbiegt". Es erfolgt also eine eigene Behandlung des NMIs, sodass die Flag fuer die Netzfrequenz beim I/O-Reset immer auf 50 Hertz gesetzt wird. Die eigene Behandlung eines gedrueckten "RESTORE" hat allerdings einen kleinen Nachteil: Befindet sich ein Steckmodul im Commodore 64, das seinen eigenen Vektor fuer die NMI-Routine besitzt, so bleibt dieser Vektor fortan unbeachtet, da es ja sein koennte, dass diese eigene Routine auch die Flag wieder auf 60 Hertz setzt.

Das Maschinenprogramm liegt im Bereich ab 49152, der vom BASIC nicht benutzt wird. Das Aendern fuer andere Bereiche duerfte jedoch nicht schwer fallen, da das Programm im Prinzip recht logisch aufgebaut ist. Wird bei der Eingabe der DATA-Zeilen ein Fehler gemacht, so wird dieser in den meisten Faellen durch das Programm bemerkt. Ist das Maschinenprogramm durch den BASIC-Traeger abgelegt worden, so kann es durch ...

SYS 49152

... aktiviert werden. Es werden der USR- und der NMI-Vektor auf eigene Routinen gesetzt sowie die Taktfrequenzflag auf 50 Hz gesetzt. Auch das Ausschalten ist moeglich. Hierbei wird der USR-Vektor wieder auf "?ILLEGAL QUANTITY ERROR" gesetzt, der NMI-Vektor erhaelt seinen Normalwert, den er beim Einschalten hat. Die Uhr laeuft natuerlich weiter (auch durch zum Beispiel Recorderoperationen wird diese nicht angehalten). Das "Abschalten" erfolgt durch ...

SYS 49253

Das Maschinenprogramm hat eine Laenge von 308 Bytes. Hier das Listing zum Eingeben ...

```
100 REM "TIME OF DAY" (IRQ-CIA) FUER DEN COMMODORE 64
110 FORI=49152TO49459:READA:X=A+X:Y=A-Y:POKEI,A:NEXT
120 IFX<>349090RY<>417THENPRINT:PRINT"CHECKSUM ERROR
130 DATA169,122,141,17,3,169,192,141,18,3,169,29,141,24,3,169,192,141,25,3,173
140 DATA14,220,9,128,141,14,220,96,72,138,72,152,72,169,127,141,13,221,172,13
150 DATA221,16,3,76,114,254,32,188,246,32,225,255,208,245,162,4,189,47,253,157
160 DATA19,3,202,208,247,162,26,189,53,253,157,25,3,202,208,247,169,127,141,13
170 DATA220,141,13,221,141,,220,169,136,141,14,220,169,8,32,179,253,76,108,254
180 DATA169,72,141,17,3,169,178,141,18,3,169,71,141,24,3,169,254,141,25,3,96
190 DATA36,13,16,108,32,130,183,192,7,208,61,173,15,220,41,127,141,15,220,160,
200 DATA169,36,32,200,192,170,208,2,169,36,201,19,144,7,248,56,233,18,216,9
210 DATA128,141,11,220,32,198,192,141,10,220,32,198,192,141,9,220,32,48,193
220 DATA141,8,220,169,,76,60,188,104,104,104,104,169,255,208,245,169,96,133,36
230 DATA32,221,192,10,10,10,10,133,37,32,221,192,5,37,197,36,176,228,96,177,34
240 DATA56,233,48,144,218,201,10,176,214,200,96,169,7,32,125,180,160,,173,11
250 DATA220,8,41,31,201,18,208,2,169,,40,16,5,248,24,105,18,216,32,31,193,173
260 DATA10,220,32,31,193,173,9,220,32,31,193,173,8,220,32,42,193,104,104,76
270 DATA202,180,72,74,74,74,74,32,42,193,104,41,15,9,48,145,98,200,96,32,221
280 DATA192,96
```

ADAPTION VON CBM-PROGRAMMEN AN DEN COMMODORE 64

Bei der Adaption von BASIC-Programmen der uebrigen Commodore-Computer muss eine Unterscheidung gemacht werden, da BASIC-Programm nicht gleich BASIC-Programm ist. Folgende Unterteilung soll hier gewaehlt werden:

1. "reine" BASIC-Programme OHNE maschinennahe Befehle wie: WAIT, POKE, PEEK, SYS, USR

2. BASIC-Programme OHNE die Befehle SYS und USR

3. kombinierte BASIC- und Maschinenprogramme mit SYS und USR

Programme des Typs 1 sind im Normalfall ohne Einschraenkungen auf den Commodore 64 uebertragbar. Einige Befehlssequenzen werden sich sogar vereinfachen lassen, speziell bei Programmen des PET 2001, da das BASIC in bestimmten Feinheiten noch modifiziert wurde. Bei Programmen des CBM 8032 wird das Bildschirmformat (auch in Hinsicht der Fensterdefinitionen) geaendert werden muessen. Ausserdem duerfen natuerlich keine BASIC 4.0-Befehle wie "CATALOG" oder aehnliches vorkommen.

Die Uebertragung von Programmen des Typs 3 wird im Normalfall nur sehr schwer moeglich sein, speziell wenn auf I/O-Bausteine zugegriffen wird. Auch die Strukturen der OS's unterscheiden sich stark von der des Commodore 64. Daher wird es meist nur dem versierten Assembler-Programmierer moeglich sein, diese Programme zu adaptieren. Benoetigte neue Systemunterlagen dazu (entsprechend dem ROM-Listing) sind bei uns staendig in Entwicklung. Fragen Sie bei Bedarf einfach nach.

Im folgenden nun Hinweise fuer die Anpassung:

Das Einladen von Programmen kann einfach durch den LOAD-Befehl erfolgen. Dies ist trotz der unterschiedlichen Anfangsadresse moeglich, da "LOAD" das Programm an den Anfang des BASIC-Bereichs verschiebt. Lediglich bei Cassettenprogrammen, die auf dem PET 2001 aufgenommen wurden, muss VOR dem Laden "POKE 43, 0" und nach dem Laden "POKE 43, 1" eingegeben werden, da hier das Abspeichern von Programmen ab Adresse 1024 vorgenommen wurde.

Die meistverwendeten systemspezifischen Befehle des BASICs 2.0 und 4.0 sollen aufgefuehrt werden:

Die Befehle "POKE 59468, 14" und "POKE 59468, 12" bewirken die Umschaltung zwischen Kleinschrift (14) und Graphik (12). Dies erfolgt beim 64er durch "PRINT CHR$ (14)" und "PRINT CHR$ (142)".

In Spielen werden oft die Adressen 151 und 152 verwendet. 151 enthaelt den Code der gedrueckten Taste, entsprechend 203 beim Commodore 64. Die Shift-Taste wird statt 152 durch 653 (zusammen mit Commodore und Control!) festgestellt. Allerdings enthaelt 151 in dem Falle, dass keine Taste gedrueckt ist nicht den Code 64 sondern vielmehr 255.

Der Bildschirm ist ein weiterer "kritischer" Bereich. Hier muessen Adressen von 32768 bis 33767 (oder bis 34767 beim 80-Zeichen-Schirm) in den Bereich von 1024 bis 2023 umgerechnet werden. Ausserdem muss die Register der Color-Nybble-Area belegt werden. Man kann sich die Arbeit jedoch wesentlich vereinfachen, wenn man den Anfang der Video-Matrix wie beim VIC-II-Chip beschrieben auf die Adresse 32768 legt.

Auch die Verwendung des Tastaturpuffers ist gebraeuchlich. Dieser belegt bei den CBM's den Bereich von 623 bis 632 (oder eventuell auch weiter). Dies entspricht den Adressen 631 bis 640 des Commodore 64. Die Anzahl gueltiger Zeichen wird in Adresse 198 (statt 158 bei den CBM's) festgelegt. Die Adresse 158 wird auch mit dem WAIT-Befehl angesprochen. Dies dient dem Programmhalt, bis eine Taste gedrueckt wird.

Das Abschalten der RUNSTOP-Taste wird durch Aendern des Inhalts von 144 erreicht. Beim BASIC 2.0 erfolgt dies durch den Wert 49 (normal 46) und beim BASIC 4.0 durch den Wert 88 (normal 85). Die entsprechenden Kommandos fuer den 64er sind "POKE 788, 52" (Abschalten) sowie "POKE 788, 49" (Einschalten).

Weitere Adressbelegungen sind:	CBMs	64er
USR-Vektor	0 - 2	784 - 786
Anfang des BASIC-Bereichs	40 - 41	43 - 44
Anfang der Variablen	42 - 43	45 - 46
Anfang der Arrays	44 - 45	47 - 48
Ende der Variablen	46 - 47	49 - 50
Ende des Arbeitsspeichers	52 - 53	55 - 56
CHRGET-Routine	112 - 135	115 - 138
IRQ-Vektor	144 - 145	788 - 789
Statusvariable ST	150	144
gedrueckte Taste	151	203
Flag fuer SHIFT	152	653
Flag fuer LOAD und VERIFY	157	10 / 147
Anzahl Zeichen im Tastaturpuffer	158	198
Flag fuer Negativdarstellung	159	199
Flag fuer Cursor ein/aus	167	204
Zeiger auf Anfang der Cursorzeile	196 - 197	209 - 210
Cursorspalte	198	211
Flag fuer QUOTE-Modus	205	212
Cursorzeile	216	214
Zaehler fuer INSERT	220	216
Tabelle der MSB's des Bildschirms	224 - 248	217 - 241
Flag fuer Kontrolle von Tape #1	249	192
Flag fuer Kontrolle von Tape #2	250	192
Tastaturpuffer	623 - 632	631 - 640
1. Cassettenpuffer	634 - 825	828 - 1019
2. Cassettenpuffer	826 - 1017	828 - 1019

SPEICHERAUFTEILUNG DES COMMODORE 64

65535 57344	RAM- BEREICH	KERNAL- ROM- BEREICH	
57343 53248	RAM- BEREICH	CHARACTER- ROM- BEREICH	BEREICH DER I/O- REGISTER
53247 49152	RAM- BEREICH		
49151 40960	RAM- BEREICH	CARTRIDGE- ROM- BEREICH	BASIC- ROM- BEREICH
40959 32768	(BASIC-) RAM- BEREICH	CARTRIDGE- ROM- BEREICH	
32767 512	(BASIC-) RAM- BEREICH		
511 256	PROZESSOR- STACK		
255 2	ZEROPAGE- RAM		
1 0	PROZESSOR- PORT		

57343 57088	I/O #2 DISC
57087 56832	I/O #1 Z80 ON
56591 56576	CIA #2 (NMI)
56335 56320	CIA #1 (IRQ)
56319 55296	COLOR-RAM- AREA
54300 54272	REGISTER SID 6581
53294 53248	REGISTER VIC 6567

MEMORY MAP DES COMMODORE 64

Speicherstellen mit Bezeichnung und dem Zusatz "unbenutzt" werden geaendert, aber nicht abgefragt oder benutzt.

Adresse	Bedeutung
0	: Datenrichtungsregister des Prozessorports
1	: I/O-Register des Prozessorports
2	: unbenutzt
3 - 4	: 45482, Vektor: FLPINT (unbenutzt)
5 - 6	: 45969, Vektor: INTFLP (unbenutzt)
7	: Suchzeichen/Trennzeichen, Ziffer beim Lesen von Zeilennummern, Register zur Ausfuehrung von AND und OR
8	: Suchzeichen/Trennzeichen, Flag fuer Anfuehrungszeichen, Register zur Ausfuehrung von AND und OR
9	: Zwischenspeicher fuer Cursorspalte bei TAB
10	: Flag fuer LOAD (=0) und VERIFY (=1) bei BASIC
11	: Laenge der einzufuegenden Programmzeile, Tabellenzeiger auf BASIC-Worte (Tokenwandlung), Ausfuehrung von AND (=0) und OR (=255), Anzahl Dimensionen bei Array-Verwaltung
12	: Flag fuer DIM
13	: Typ des arithmetischen Ausdrucks (String=255, numerisch=0)
14	: Art der numerischen Variablen (Integer=128, real=0)
15	: Flag fuer DATA (Umwandlung in Tokens), Flag fuer Anfuehrungszeichen (LIST), Flag fuer GARBAGE COLLECT bei Stringeinbau
16	: Flag zum Sperren der Annahme von Integer- und Feldvariablen (FN und FOR)
17	: Flag fuer INPUT (=0), GET (=64) und READ (=152)
18	: Speicher fuer Operatormaske ("<=>"), Vorzeichenflag bei trigonometrischen Funktionen
19	: Flag fuer direkte Eingabe/Ausgabe (=0) oder ueber Filekommando (=Filenummer)
20 - 21	: Zeilennummern und Integerwerte (Adressen)
22	: Zeiger auf Tabelle der Stringdescriptoren
23 - 24	: Zeiger auf zuletzt benutzten Stringdescriptor
25 - 33	: Tabelle der Stringdescriptoren (String-Stack)
34	: Zeiger in Arrayheader, Zwischenspeicher fuer Operator
34 - 35	: Uebertragungszeiger, Lesen von Zeilennummern, Zeiger fuer Speicherverschiebungen, Suchzeiger, Variablenzeiger, Stringzeiger, Sprungvektor
36	: Zwischenspeicher bei NEXT
36 - 37	: Zeiger fuer indirekte Speicherung
40	: Laenge eines Feldelementeintrags
40 - 41	: Faktor fuer Feldelementberechnung in Array
38 - 41	: Ergebnisbereich der Mantisse ("*" und "/")
42	: Rundungsstelle bei Division (unbenutzt)
43 - 44	: Zeiger auf Start des BASIC-Programms
45 - 46	: Zeiger auf Start der nichtindizierten Variablen
47 - 48	: Zeiger auf Start der Feldvariablen
49 - 50	: Zeiger auf Ende (plus eins) der Feldvariablen
51 - 52	: Zeiger auf Anfang des Stringbereichs
53 - 54	: Zeiger fuer Stringuebertragungen
55 - 56	: Zeiger auf Ende (plus eins) des BASIC-Speichers
57 - 58	: momentane BASIC-Zeilennummer
58	: Flag fuer Direktmodus (=255)
59	: gleich null, falls kein CONT moeglich
59 - 60	: Zeilennummer fuer CONT (bei Abbruch)
61 - 62	: CHRGET-Pointer auf Befehl (bei CONT)

63 - 64: Zeilennummer der momentanen DATA-Zeile
65 - 66: Zeiger auf naechstes Element fuer DATA
67 - 68: Zeiger auf Eingabequelle zur Auswertung (INPUT, GET, READ)
69 - 70: Variablenname
71: Zwischenspeicherung (YR) fuer Tabellenzeiger (Stringerzeugung aus Fliesskomma oder Uhrzeit)
71 - 72: Zeiger auf Variable
73 : Zwischenspeicher fuer Geraetenummer beim Lesen der LOAD/SAVE/VERIFY-Parameter, Zwischenspeicher fuer Filenummer beim Lesen der Fileparameter fuer OPEN und CLOSE, Zwischenspeicher fuer zweiten WAIT-Parameter
74 : Zwischenspeicher fuer Geraetenummer beim Lesen der Fileparameter fuer OPEN und CLOSE, Zwischenspeicher fuer dritten WAIT-Parameter
73 - 74: Zeiger auf FOR-NEXT-Variable (fuer Stacksuche), Zeiger auf Variable bei Zuweisungen (INPUT, GET, LET, READ)
75 : Zeiger auf Prioritaetsflag in Tabelle, Flag fuer fehlenden Operator bei Auswertung
75 - 76: Zwischenspeicher fuer CHRGET-Pointer (INPUT, GET, READ)
77 : Maske fuer Vergleichsoperationen (Auswertung)
78 - 79: Function-Variablenzeiger fuer DEF und FN, Zeiger in Stringdescriptor bei GARBAGE COLLECT
80 - 81: Zeiger auf Stringdescriptor bei LEFT$, RIGHT$, MID$, LET, Stringaddition und Transfer von Strings in den Stringbereich
78 - 82: Fliesskommaregister (Zwischenspeicherung) (Exponent bei Potenzierung)
83 : Schrittweite fuer GARBAGE COLLECT
84 : Code 76 fuer "JMP"
85 : Zwischenspeicher fuer Schrittweite (GARBAGE COLLECT), Zwischenspeicher der Ruecksprungadresse high (MID$)
86 : Zwischenspeicher fuer Rundungsstelle von FAC (Addition und EXP)
85 - 86: Sprungvektor fuer Funktionen
88 - 89: Pointer fuer Blocktransfer (Programmzeilen, Variablen, Arrays), Pointer in Array fuer GARBAGE COLLECT
90 - 91: Pointer fuer Blocktransfer (Variablen, Arrays, Programmzeilen, Strings bei GARBAGE COLLECT)
87 - 91: Fliesskommaregister (Zwischenspeicherung) (Polynomauswertung, TAN)
93 : Anzahl Stellen fuer TI$, Bitzaehler fuer 16-Bit-Binaermultiplikation, Zaehler fuer Anzahl Nachkommastellen (STRFAC), Dezimalexponent (FACSTR)
94 : Flag fuer Exponentialdarstellung (FACSTR), Dezimalexponent (STRFAC),
95 : Flag fuer Dezimalpunkt (STRFAC)
96 : Vorzeichen des Exponenten (STRFAC)
95 - 96: Pointer fuer Blocktransfer, Startadresse einer Programmzeile, Pointer fuer Verwaltung von Variablen, Pointer fuer Stringtransfer (GARBAGE COLLECT)
92 - 96: Fliesskommaregister (Zwischenspeicherung) (Polynomauswertung)
97 : Exponent von FAC
97 - 99: Stringdescriptor (Stringlaenge, Stringpointer)
101 : Argument zu ON (Ergebnis von GETBYT), Stringlaenge fuer MID$

```
100 -  101: Variablenpointer (Elementauswertung)
            Arrayindex (Feldelementberechnung),
 98 -  101: Mantisse von FAC (MSB bis LSB, 32 Bits)
102       : Vorzeichen von FAC
103       : Vorzeichen fuer STRFAC,
            Zaehler fuer Polynomgrad (Polynomauswertung)
104       : Vorzeichen (Umwandlung FAC in Integerformat)
105       : Exponent von ARG
108 -  109: Pointer auf zweiten String fuer Stringvergleich
106 -  109: Mantisse von ARG (MSB bis LSB, 32 Bits)
110       : Vorzeichen von ARG
111       : verknuepftes Vorzeichen von FAC und ARG
            (Flag fuer gleiches Vorzeichen FAC und ARG),
112       : Rundungsstelle von FAC (fuenftes Mantissenbyte)
111 -  112: Stringdescriptorzeiger fuer Uebertragung des
            Strings in den Stringbereich,
            Pointer auf Descriptor des ersten Strings
            (Stringaddition)
113       : Zwischenspeicher fuer Pointer auf codierte
            Zeile (Umwandlung von Text in Tokens),
            Stellenzaehler fuer Zuweisung an TI$,
            Pointer auf erzeugten String (FACSTR)
113 -  114: Pointer auf Tabelle mit Koeffizienten
            (Polynomauswertung),
            Pointer fuer Arrayhandling (Elementlaenge,
            Feldlaenge, Positionsberechnung fuer Element),
            Zwischenspeicher fuer CHRGET-Pointer bei VAL,
            Endpointer fuer Uebertragung von Strings in den
            Stringbereich
115 -  138: CHRGET-Routine, liest Zeichen aus BASIC-Text
122 -  123: CHRGET-Pointer, zeigt in BASIC-Text;
            wird auch zum Auswerten von Eingaben und
            aehnlichem auf andere Quellen umgesetzt
            (INPUT, GET, READ, FN, VAL)
139 -  143: letzte Zufallszahl bei RND
144       : I/O-Statusbyte fuer Cassette und seriellen Bus
145       : Zwischenspeicher Tastaturausgang fuer RUNSTOP
            und Abfrage der Commodore-Taste bei Tape-Read
```

Die Angaben "Read" und "Write" beziehen sich auf die Verwendung dieser Adressen bei den Recorderoperation!

```
146       : Korrekturflag zur Anpassung der Timing-
            Konstanten bei Gleichlaufschwankungen (Read)
147       : Flag fuer LOAD (=0) und VERIFY (=1) fuer KERNAL
148       : Flag fuer "Zeichen gepuffert" bei seriellem Bus
149       : serielles Register fuer Ausgabe von Daten
            (serieller Bus)
            sowie Puffer fuer letztes Byte (EOI)
150       : Flag fuer "End of Block empfangen" (Read)
151       : Zwischenspeicher fuer Indexregister beim Holen
            von einzelnen Zeichen (GETIN)
152       : Anzahl an geoeffneten logischen Files
            (Zeiger in Tabelle der Fileparameter)
153       : aktives Eingabegeraet, normalerweise Tastatur
154       : aktives Ausgabegeraet, normalerweise Bildschirm
155       : Register zur Bestimmung des Paritybits von
            Datenbytes (Read/Write)
156       : Flag fuer "Byte empfangen" (Read)
157       : Flag fuer Ausgabemodus fuer Betriebssystem-
            meldungen (siehe Beschreibung Systemroutinen),
            Festlegung fuer Direktmodus und Programmodus
```

158   : Zwischenspeicher fuer Kennzeichnung des Dateityps, Pointer auf Filenamen bei Uebertragung des Filenamens (beides beim Erzeugen des Dateikopfs fuer Tape-Write),
Zaehler fuer Anzahl Lesefehler mal zwei fuer Pointer in Adressentabelle im Stack fuer Fehlerkorrektur (Read)

159   : Pointer auf Filenamen in Dateikopf bei Uebertragung des Filenamens (Erzeugung des Dateikopfs fuer Tape-Write),
Korrekturzaehler fuer Pass2 beim Lesen von Band (Pointer auf Adressentabelle im Stack, Read)

160 - 162: interne (softwaremaessig realisierte) Uhr (TI, TI$, Reihenfolge MSB bis LSB!)

163   : EOI-Flag bei Ausgabe auf seriellen Bus, Bitzaehler fuer Lesen von Datenbytes (Read)

164   : serielles Register fuer Empfang von Daten vom seriellen Bus, Flag fuer Empfang/Ausgabe beider Impulse, die ein Datenbit festlegen, Impulszaehlung (Read/Write)

165   : Zaehler fuer acht Bits bei Ausgabe/Empfang von Daten ueber seriellen Bus, Flag fuer EOI (=0) und Zeitfehler (=1) bei Timeout (serieller Bus)
Zaehler fuer Synchronisationsbytes (Zaehlung, Write)

166   : Pointer in Cassettenpuffer fuer Filehandling

167   : Anzahl noch zu lesende Blocks (Read),
Zaehler fuer Dauer der Shorts (Write),
Zwischenspeicher fuer empfangenes Bit (RS-232)

168   : Flag fuer Lesefehler eines Bits und Parityerror (Read),
Flag fuer "Byte-Impuls geschrieben" (Write),
Bitzaehler fuer Empfang von Bytes (RS-232)

169   : Flag fuer Impulslaengenwechsel zur Erkennung von Bit-Lesefehlern (Read),
Flag fuer "Long-Impuls (nach "Byte"-Impuls) geschrieben" (Write),
Flag fuer Empfang des Startbits (RS-232)

170   : Flag fuer Synchronisationszaehlung (=1 bis 9), Abtastung (=0), Lesen (=64) und Ende (=128),
serielles Shiftregister fuer Empfang von Daten (RS-232)

171   : Register zur Errechnung der Puffer-Pruefsumme (Checksum, Read), Laenge des Headers (Write),
Register zur Errechnung der Parity beim Empfang von Daten (RS-232)

172 - 173: Transportzeiger fuer Tape und SAVE auf seriellem Bus, Pointer fuer Scrolling (Videomatrix, Quellzeile)

174 - 175: Endadresse fuer Write/SAVE (Programm), Pointer fuer Scrolling (Colornybbles, Quellzeile),
Transportzeiger fuer LOAD von Diskette

176   : Timing-Konstante zur Anpassung der Lesegeschwindigkeit, Korrektur durch (146) (Read)

177   : vergangene Zeit seit letzter negativer Flanke (Read)

178 - 179: Pointer auf Startadresse des Recorderpuffers

180   : Freigabeflag fuer TimerA (Tape Read),
Bitzaehlelr fuer Ausgabe von Bytes (RS-232)

181   : Flag fuer "gueltiges EOB empfangen" (Read),
naechstes zu sendendes Bit (RS-232)

182   : Flag fuer Lesefehler des Bytes (Tape Read) und Vergleichsfehler (VERIFY);
Flag fuer "Block geschrieben" (Tape Write),
serielles Shiftregister zur Ausgabe von Daten (RS-232)

```
183         : Laenge des Filenamens
184         : aktuelle logische Filenummer
185         : aktuelle Sekundaeradresse
186         : aktuelle Geraetenummer
187 -   188: Pointer auf Filenamen
189         : Register fuer gelesenes Byte sowie Checksum
              beim Lesen (letztes gelesenes Byte, Read),
              serielles Bit-Shiftregister (Write),
              Register zur Bestimmung der Parity bei der
              Ausgabe von Daten (RS-232)
190         : Zaehler fuer Anzahl noch zu verarbeitender
              Blocks (Read),
              Anzahl noch zu schreibende Blocks (Write)
191         : serielles Shiftregister zum Lesen von Bytes
              (Tape Read)
192         : Motorkontrolle zum Anhalten des Recordermotors
193 -   194: Pufferstartadresse, Programmstartadresse
              (SAVE, Tape Read/Write), Pointer fuer RAM-Test
195 -   196: Startadresse (Appendadresse) bei LOAD (KERNAL),
              Uebertragungszeiger fuer Vektortabelle
197         : Tastaturmatrixcode aus vorherigem Aufruf der
              Tastaturabfrage (SCNKEY)
198         : Anzahl gueltige Zeichen im Tastaturpuffer
199         : Flag fuer Negativdarstellung (RVS)
200         : Pointer auf letztes Zeichen (ungleich Space) in
              Bildschirmzeile beim Lesen der Eingabezeile
201         : Cursorzeile bei Aufruf von CHRIN,
              Flag fuer Cursorzeilenwechsel bei Eingabe
202         : Cursorspalte bei Aufruf von CHRIN
203         : Tastaturmatrixcode aus SCNKEY-Aufruf,
              Pointer in Decodierungstabelle fuer Tastatur
204         : Flag fuer Cursor ein (=0) und aus
205         : Zaehler fuer Cursorblinkdauer
206         : Zeichen unter Cursor, falls Cursor hell
207         : Flag fuer Cursor hell/dunkel (wenn (204) = 0)
208         : Flag fuer letztes Zeichen beim Lesen von
              Zeichen vom Bildschirm, Flag fuer "RETURN"
209 -   210: Pointer auf Anfang der Cursorzeile im
              Bildschirm-RAM, Zielpointer fuer Scroll
211         : Cursorspalte innerhalb der Cursorzeile,
              Indirect-Pointer zu (209/210)
212         : Flag fuer Anfuehrungszeichen (Quote-Modus)
213         : Laenge der aktuellen Cursorzeile (39 oder 79)
214         : Nummer der Cursorzeile (0 bis 24)
215         : Register fuer gelesenes Bit (Tape Read),
              Puffer-Pruefsumme (Tape Write),
              Zwischenspeicher fuer Zeichencode sowohl bei
              Umwandlung von ASCII in Bildschirmcode als
              auch umgekehrt (bei Ausgabe von Zeichen und
              beim Lesen von Zeichen vom Bildschirm)
216         : Insertzaehler, Anzahl noch ausstehender Inserts
217 -   241: Tabelle der Doppelzeilenkennzeichnungen,
              Bit 0 und Bit 1 enthalten die Page# innerhalb
              der Videomatrix; ist Bit 7 geloescht, so
              handelt es sich um eine Fortsetzungszeile
242         : Einfachzeilenkennzeichnung der "26. Zeile"
              (falls auf Fortsetzungszeile zur letzten Bild-
              schirmzeile zugegriffen wird)
243 -   244: Pointer auf Anfang der Cursorzeile im
              RAM der Colornybbles, Zielpointer der Farbcodes
              fuer Scroll
245 -   246: Pointer auf Decodierungstabelle fuer Tastatur-
              abfrage (SCNKEY)
247 -   248: Zeiger auf Beginn des Empfangspuffers (RS-232)
249 -   250: Zeiger auf Beginn des Sendepuffers (RS-232)
251 -   254: unbenutzt
```

255 - 271: Puffer fuer Zahlenumwandlungen von Fliesskomma in Strings sowie fuer Erzeugung von TI$
256 - 379: Puffer fuer Fehlerkorrektur beim Lesen von Band (enthaelt Adressen der fehlerhaften Bytes)
256 - 511: Hardwarestack des Prozessors, wird ausser der normalen Funktion des Stacks auch zur Speicherung von Daten fuer GOSUB und FOR-NEXT-Schleifen verwendet
508 - 511: Speicher fuer Zeilennummer und Pseudolink fuer Einbau von Programmzeilen in BASIC-Text
512 - 600: BASIC-Eingabepuffer, wird verwendet fuer INPUT, GET (jeweils auch fuer Filehandling), Puffer fuer Eingabe von Zeilen, Umwandlung in Tokens
601 - 610: Tabelle der Filenummern
611 - 620: Tabelle der Geraetenummern
621 - 630: Tabelle der Sekundaeradressen
631 - 640: Tastaturpuffer
641 - 642: MEMBOT, Speicherbeginn fuer BASIC
643 - 644: MEMTOP, Speicherende fuer BASIC
645 : Timeout-Flag fuer seriellen Bus (unbenutzt)
646 : Cursorfarbe (Druckfarbe fuer Zeichenausgabe)
647 : Farbe unter Cursor, falls Cursor hell
648 : Startpage der Videomatrix
649 : maximale Groesse des Tastaturpuffers (maximale Anzahl Zeichen im Tastaturpuffer)
650 : Flag fuer REPEAT:
Bit 7 gesetzt: REPEAT fuer alle Tasten
Bit 6 gesetzt: REPEAT fuer keine Taste
sonst nur fuer Cursorsteuerung und SPACE
Bit 7 hat Prioritaet vor Bit 6
651 : Zaehler der Wiederholungszeit fuer REPEAT (Geschwindigkeit)
652 : Zahler der Anspruchszeit fuer REPEAT (Verzoegerung)
653 : Flag fuer Kombinationstasten:
Bit 0: Shift
Bit 1: Commodore
Bit 2: Control
das entsprechende Bit ist gesetzt, wenn die Taste gedrueckt ist
654 : Kopie von (653) fuer Abfrage, ob Commodore- und Shift-Taste schon beim letzten Aufruf gedrueckt waren (fuer Umschaltung von Graphik und Text)
655 - 656: Sprungvektor fuer Tastaturabfrage (kann zum Beispiel verwendet werden, um die Funktionstasten zu belegen)
657 : Flag fuer Blockierung der Umschaltung von Text und Graphik durch die Shift- und Commodore-Taste (durch CHR$(8) und CHR$(9))
658 : Insert Enable, dient zur Erkennung, ob beim Ueberschreiten einer Einfachzeile eine Leerzeile eingefuegt (Editiermodus) oder die Zeile ueberschrieben werden soll (Ausgabe von Zeichen durch Programm (PRINT))
659 : Kontrollregister (RS-232)
660 : Kommandoregister (RS-232)
661 - 662: Wert fuer Baud-Rate aus Tabelle (RS-232)
663 : RS-232 Statusbyte
664 : Wortlaenge (RS-232)
665 - 666: Wert fuer Timer beim Senden (RS-232)
667 : Zeiger auf Ende des Empfangspuffers
668 : Zeiger auf naechstes Zeichen im RS-232-Empfangspuffer
669 : Zeiger auf zu uebertragendes Byte im Sendepuffer

```
 670       : Zeiger auf naechste freie Stelle im Sendepuffer
 671 -  672: Zwischenspeicher des IRQ-Vektors waehrend
           Cassettenoperationen
 673       : Flagregister fuer aktive RS-232-NMIs
 674       : Wert fuer CRA (IRQ-CIA) fuer Neustart von
           Timer A beim Lesen von Cassette
 675       : Wert des Interrupt Flag Registers (Tape Read)
 676       : Flag fuer "Timer A abgelaufen"
           (Underflow, Tape Read)
 677       : Nummer der Fortsetzungszeile beim Erweitern
           einer Einfachzeile (Scroll)
 678       : Flag fuer Quarzfrequenz der verschiedenen
           Versionen des Commodore 64
 679 -  767: unbenutzt
 768 -  769: 58251, Vektor: Ausgabe von Fehlermeldungen
 770 -  771: 42115, Vektor: Eingabewarteschleife nach READY.
 772 -  773: 42364, Vektor: Umwandlung Klartext in Tokens
 774 -  775: 42778, Vektor: Umwandlung Tokens in Klartext
 776 -  777: 42980, Vektor: Routinenaufruf (Interpreter)
 778 -  779: 44678, Vektor: Elementauswertung (FRMEVL)
 780       : Uebergabewert bei SYS-Befehl (Accu)
 781       : Uebergabewert bei SYS-Befehl (XR)
 782       : Uebergabewert bei SYS-Befehl (YR)
 783       : Uebergabewert bei SYS-Befehl (Statusregister)
 784       : Code 76 fuer "JMP"
 785 -  786: 45640, USR-Vektor
 787       : unbenutzt
 788 -  789: 59953, IRQ-Vektor
 790 -  791: 65126, BRK-Vektor
 792 -  793: 65095, NMI-Vektor
 794 -  795: 62282, OPEN-Vektor
 796 -  797: 62097, CLOSE-Vektor
 798 -  799: 61966, CHKIN-Vektor
 800 -  801: 62032, CHKOUT-Vektor
 802 -  803: 62259, CLRCHN-Vektor
 804 -  805: 61783, CHRIN-Vektor
 806 -  807: 61898, CHROUT-Vektor
 808 -  809: 63213, STOP-Vektor
 810 -  811: 61758, GETIN-Vektor
 812 -  813: 62255, CLALL-Vektor
 814 -  815: 65126, unbenutzt
 816 -  817: 62629, LOAD-Vektor
 818 -  819: 62957, SAVE-Vektor
 820 -  827: unbenutzt
 828 - 1019: Cassettenpuffer
1020 - 1023: unbenutzt
```

Unterschiede in der Speicherbelegung des VIC-20:

Der USR-Vektor (zusammen mit JMP-Code) belegt die Adressen 0 bis 2 anstelle der Adressen 784 bis 786, der Prozessorport entfaellt.

Adressen 673 bis 678 sind beim VIC-20 unbenutzt.

Die Tabelle der Doppelzeilenkennzeichnungen belegt nur den Bereich von 217 bis 239 (da nur 23 Bildschirmzeilen). Adresse 240 erfuellt dadurch die Funktion der Adresse 242 des Commodore 64. Adresse 241 wird (wie 243 beim Commodore 64) beim Loeschen des Bildschirms gesetzt, jedoch in keinem Zusammenhang damit benutzt. Vermutlich sollte der Speicherungsbefehl an die Adressen fuer die "24." bzw. "26." Zeile gehen, diese werden jedoch schon vorher initialisiert. Adresse 242 stand beim VIC-20 noch frei und hat die Funktion von (677) des Commodore 64.

Im Gegensatz zu den vorherigen Computermodellen von Commodore verfuegt der Commodore 64 nicht ueber eine 6502 als Prozessor, sondern eine 6510. Auf die Unterschiede und Besonderheiten, die sich daraus ergeben, soll im folgenden eingegangen werden.

Diese 6510 verfuegt ueber den gleichen Befehlssatz wie die 6502 auch und ist daher vollkommen softwarekompatibel. Die Taktfrequenz betraegt nicht genau ein Megahertz - wie zum Beispiel bei den CBMs - sondern etwas weniger (oder bei den Geraeten, die fuer die Fernsehnorm NTSC ausgelegt sind, etwas mehr) als ein MHz. Die genauen Angaben finden sich bei der Erklaerung der Systemroutinen. Der einzige wirkliche Unterschied besteht darin, dass sie zusaetzlich ueber einen I/O-Port, bestehend aus sechs Leitungen, veruegt. Jede Leitung dieses Ports laesst sich getrennt als Eingang oder Ausgang schalten. Dieser Port wird beim Commodore 64 zur Steuerung des Recorders und der Speicherverwaltung verwendet. Das besondere an diesem I/O-Port (Prozessorport genannt) ist, dass sich Datenrichtungsregister und Ausgaberegister in der Zero-Page befinden:

Adresse 0: DATA DIRECTION REGISTER
Adresse 1: OUTPUT REGISTER

Diese beiden Speicherstellen stehen daher nicht mehr fuer die Datenspeicherung zur Verfuegung (weswegen auch der USR-Vektor verlegt werden musste). Allerdings koennen nun sechs Steuerleitungen durch die Zero-Page-Adressierung erreicht werden. Diese Steuerleitungen werden ueber die Bits 0 bis 5 (Bits 6 und 7 sind als Portleitungen nicht vorhanden) kontrolliert und haben folgende Bedeutungen:

Bit 0: (Output) LORAM, Kontrolleitung fuer Speicherbelegung
Bit 1: (Output) HIRAM, Kontrolleitung fuer Speicherbelegung
Bit 2: (Output) CHAREN, Ein-/Ausblenden des Zeichengenerator
Bit 3: (Output) Schreibleitung fuer Recorderoperationen
Bit 4: (Input) Feststellung, ob Recordertaste gedrueckt ist
Bit 5: (Output) Ein- und Ausschalten des Recordermotors

Die Kontrolle ueber die Funktion einer Datenleitung als Eingang oder Ausgang erfolgt wie in allen anderen Faellen auch: Das entsprechende Bit in Adresse 0 (Datenrichtungsregister) muss gesetzt werden, wenn diese Portleitung als Ausgang programmiert werden soll. Die Funktion als Eingang erhaelt die Leitung, wenn das zugehoerige Bit des Datenrichtungsregisters geloescht ist. Die Umprogrammierung ist jedoch beim Commodore 64 meist sinnlos, da der gesamte Prozessorport bereits durch das Betriebssystem und durch die Hardware festgelegte Funktionen erfuellt.

Die Bits 3 bis 5 werden fuer Recorderoperationen verwendet. Bit 3 dient dazu, Impulse auf Band zu schreiben (siehe ROM-Listing, Adresse 64422 ff). Bit 4 ist geloescht, wenn eine Recordertaste (ausser REC und STOP) gedrueckt ist, ansonsten gesetzt (auch, wenn kein Recorder angeschlossen ist). Bit 5 dient der Recordermotorkontrolle. Soll der Motor eingeschaltet werden, so muss dieses Bit geloescht werden. Das Anhalten erfolgt entsprechend durch Setzen von Bit 5. Erfolgt dies allerdings bei zugelassenem Interrupt (also zum Beispiel im Normalfall von BASIC aus), so muss ausserdem Adresse 192 auf einen Wert ungleich null gesetzt werden, da ansonsten bei gedrueckter PLAY-Taste der Recordermotor nicht angehalten werden kann.

Das Speicherverwaltungskonzept des Commodore 64

Der Commodore 64 verfuegt, was in der Werbung auch gross angepriesen wird, ueber 64 Kilobytes an RAM, 20 Kilobytes ROM und ansonsten auch noch ueber einige Bausteine, die gesonderte Funktionen uebernehmen. Nur hat dies alles einen Haken, der jedem, der sich schon ein wenig mit der Architektur der 65xx CPUs beschaeftigt hat, sofort auffallen muesste: ein Prozessor mit einem Adressbus von sechzehn Bits kann nur 64 Kilobytes adressieren. Egal, wie man es dreht und wendet, irgendwo ist etwas zuviel. Und fuer denjenigen, der "nur" in reinem BASIC (ohne maschinennahe Befehle wie POKE oder PEEK) programmiert, sind dies leider 26 KB, die von den versprochenen 64 KB fehlen, da beim Einschalten nur 38 KB als frei verfuegbar gemeldet werden. Von den 64 KB gehen natuerlich erst einmal 2 KB fuer Bildschirm (von 1024 bis 2047) und Systemspeicher (von 0 bis 1023) ab. Ausserdem stehen, wie dies aus Speicheruebersichten bekannt ist, noch weitere 4 KB im Bereich von 49152 bis 53247 zur Verfuegung, von denen man als BASIC-Programmierer zwar selten etwas hat, aber man weiss immerhin, dass sie vorhanden sind. Bleiben also 20 KB, was genau dem Umfang des uebrigen Speichers entspricht. Dieser ist zum Beispiel von zwei Betriebssystem-ROMs zu je 8 KB belegt (auch hier "fehlen" 4 KB an den 20 KB ROM). Die restlichen 4 KB des adressierbaren Speicherbereichs (von 53248 bis 57343) der 6510 wird vom I/O-Bereich, bei dem - wie bisher immer - sehr grosszuegig mit der Adressdecodierung vorgegangen wurde, sodass saemtliche Register sich ueber einige Adressen ansprechen lassen, belegt.

Die restlichen 20 KB an RAM liegen "parallel" oder "unter" dem Bereich, der von I/O und ROM eingenommen wird. Fuer BASIC-Programme oder Variablen laesst sich dieser Bereich normalerweise nicht nutzen, jedoch durchaus fuer zum Beispiel Daten von hochaufloesenden Graphiken. Das Problem besteht nur darin: Wie erreicht man diesen RAM-Bereich? Dazu erst einmal eine kurze Zusammenfassung:

Die 64 KB an RAM fuellen den gesamten Adressbereich des Prozessors von 0 bis 65535. Da jedoch beim Einschalten des Geraets BASIC vorhanden sein muss, werden zwei BASIC-ROMs (im Bereich von 40960 bis 49151 und 57344 bis 65535) "ueber" das RAM gelegt, sodass fuer den Prozessor dieses RAM nicht mehr existiert, sondern nur noch der ROM. Ausserdem befinden sich im Adressbereich von 53248 bis 57343 die Register von Bausteinen, die fuer die Kommunikation mit der Aussenwelt sorgen. Auch das RAM in diesem Gebiet ist fuer den Prozessor nicht vorhanden. Lediglich im Bereich von 2 bis 40959 (sowie von 49152 bis 53247) ist RAM benutzbar.

Es muss jedoch Moeglichkeiten geben, sich diesen Bereich nutzbar zu machen. Fuer die Steuerung der Bereichsverteilung existieren daher (von einer Anzahl weiterer Leitungen, die fuer den Programmierer jedoch nicht erreichbar sind, abgesehen) hauptsaechlich fuenf Kontrolleitungen, die den Speicheraufbau regeln. Dies sind folgende ...

LORAM : Bit 0 des Prozessorports
HIRAM : Bit 1 des Prozessorports
CHAREN: Bit 2 des Prozessorports
EXROM : Anschluss 9 des Cartridge Expansion Ports
GAME : Anschluss 8 des Cartridge Expansion Ports

Wie ersichtlich, sind nur die ersten drei Leitungen softwaremaessig zu aendern, die letzteren dienen der Kontrolle von Einschubmodulen wie BASIC-Erweiterungen sowie Spielen, die zum Beispiel fuer den Ultimax (VC-10) gedacht sind.

Alle fuenf Leitungen sind 'active low', sodass ihnen der Wert null zur Ausuebung ihrer Funktion zugewiesen werden muss (entsprechend gegen Masse gelegt).

Zuerst soll auf die alleinige Wirkung von LORAM in Form eines Beispiels eingegangen werden. LORAM steuert den Bereich von 40960 bis 49151. Dieser enthaelt im Normalfall den BASIC-ROM. Allerdings ist dieser ROM, wie andere Bereiche auch, von RAM "unterlegt". Wird LORAM jetzt durch ...

```
POKE 1, PEEK (1) AND 254
```

... der Wert null zugewiesen, so wird dieses ROM durch den entsprechenden RAM-Bereich ausgetauscht. Somit befindet sich nun das BASIC-ROM nicht mehr in dem durch den Prozessor erreichbaren Adressbereich, dafuer ist nun RAM vorhanden. Der obige Befehl darf natuerlich nicht einfach so eingegeben werden. Schliesslich benoetigt man ja den BASIC-ROM. Wird dies trotzdem getan, so saegt man vergleichsweise an dem Ast, auf dem man selbst sitzt, da dieser Befehl zu einem Absturz des Computers fuehrt, im guenstigsten Fall wird ein BASIC-Warmstart ausgefuehrt.

Nun eine weitere Eigenschaft dieses Ueberlappungsprinzips: jegliche SCHREIB-Befehle, die in einen Bereich zielen, in dem sich ROM befindet, werden in den fuer den Prozessor nicht lesbaren RAM-Bereich "umgelenkt". Wird also ...

```
POKE 40960, 255
```

... eingegeben, so wird dieser POKE-Befehl nicht an den BASIC-ROM weitergeleitet. Dies haette auch wenig Sinn, da sich ein ROM ja gerade durch die Eigenschaft auszeichnet, seinen Inhalt beizubehalten. Stattdessen wird nun die Adresse 40960 des sich "unter" dem ROM befindlichen RAMs mit dem Wert 255 beschrieben. Man kann daher den gesamten RAM-Bereich, "ueber" dem sich ROM befindet (also weitere 16 KB der noch fehlenden 20 KB an RAM), aendern, jedoch (noch) nicht lesen. Somit hat die folgende Befehlszeile ...

```
FOR I = 40960 TO 49151 : POKE I, PEEK (I) : NEXT
```

... durchaus einen Sinn, wenn es auch auf den ersten Blick sinnlos erscheint, in eine Adresse den Wert zu schreiben, der sich doch sowieso schon darin befindet. Inzwischen wissen Sie jedoch, dass durch diese Befehlszeile der Inhalt des BASIC-ROMs, das sich momentan in diesem Bereich befindet, in das "darunterliegende" RAM kopiert wird. ROM und RAM dieses Bereichs sind nun also identisch (allerdings sollte darauf hingewiesen werden, dass gerade im Umgang mit Peripheriebausteinen es vorkommen kann, dass ein Wert in ein Register gespeichert wird, der sich schon darin befindet). Jetzt hindert uns auch nichts mehr daran, das BASIC-ROM durch das RAM auszutauschen. Geben Sie doch nun doch einmal "POKE 1, PEEK (1) AND 254" ein. Auch wenn sich scheinbar nichts aendert, so ist nun das BASIC-ROM fuer den Prozessor nicht mehr vorhanden. Alle Daten, die sonst aus dem BASIC-ROM geholt werden, alle Programmsequenzen, die normalerweise im BASIC-ROM ablaufen, all dies geschieht nun im RAM, das nun ueber den BASIC-ROM hinueber gehoben wurde (beim Ausprobieren sollten uebrigens eventuelle Erweiterungen entfernt werden).

Was laesst sich nun damit anfangen? Haben Sie sich noch nicht ueberlegt, was man im Betriebssystem alles aendern koennte. Geben Sie doch einfach einmal ...

POKE 41853, 32

... ein. Das Wort "READY." erscheint ploetzlich ohne den Punkt (ueber den Sinn dieser Aenderung kann man natuerlich geteilter Meinung sein). Schauen Sie am besten einmal im ROM-Listing nach. Es koennen nun deutsche Fehlermeldungen kreiert werden, sogar deutsche Befehlsworte sind theoretisch moeglich (aber wohl nicht unbedingt sinnvoll). Auch das Programm fuer die Abfrage der Echtzeituhr (siehe Beschreibung der CIA, Time of Day) kann in den Bereich fuer BASIC direkt hineingeschrieben werden. Allerdings sollten nicht solche Teile geaendert werden, die waehrend des Aenderungsvorgangs aufgerufen werden. Dazu sollte man erst zurueckschalten (auch durch RUNSTOP/RESTORE moeglich).

Beim Commodore 64 existieren nun noch weitere drei Bereiche, die verschiedene Inhalte haben koennen, je nach Zustand der Kontrolleitungen. Dies sind folgende:

Der Bereich von 57344 bis 65535 enthaelt im Normalfall den KERNAL-ROM (dient zur Bearbeitung von hardwareabhaengigen Dingen). Die zweite Moeglichkeit fuer diesen Bereich ist die Belegung dieses Bereichs mit RAM.

Der zweite aenderbare Bereich ist der von 53248 bis 57343. Hier existieren gar drei Moeglichkeiten. Zuerst ist dies die Verwendung dieses Bereichs als I/O-Bereich. Hier befinden sich die Register der CIAs, des VIC-II-Chips und des Soundbausteins. Die zweite Moeglichkeit besteht im Einblenden des Zeichengenerators, was Ihnen vielleicht aus dem Kapitel der Zeichendefinierung noch bekannt ist. Der Zeichengenerator, normalerweise fuer den Prozessor nicht erreichbar, kann den I/O-Bereich ersetzen und dann (fuer Aenderungen) ausgelesen werden. Als letzte Moeglichkeit existiert natuerlich auch hier die der Belegung dieses Bereichs mit RAM.

Zum dritten Bereich muss nicht viel gesagt werden. Es ist der Adressbereich von 40960 bis 49151, der zuvor als Beispiel diente. Er enthaelt entweder den BASIC-ROM, natuerlich RAM oder auch den Inhalt eines Cartridges, das in den Erweiterungsport gesteckt wurde.

Ausserdem existiert noch der Bereich von 32768 bis 40959: Er enthaelt, wenn kein Erweiterungsmodul gesteckt wurde, RAM, das von BASIC aus verwendet werden kann. Ausserdem kann es den Inhalt eines Cartridge-ROMs enthalten (zum Beispiel EXBASIC), wodurch aber acht Kilobytes an RAM nicht mehr verfuegbar sind.

Nun eine genau Uebersicht, welche Leitungen welche Zustaende einnehmen muessen, um eine gewuenschte Speicherbelegung zu erreichen (nicht alle theoretisch moeglichen Kombinationen sind auch erreichbar). Die Leitung CHAREN wird hier ausser acht gelassen, sie wird anschliessend erlaeutert.

```
57344-65535: RAM   : HIRAM = 0
             KERNAL: HIRAM = 1
```

Soll also der Bereich der oberen 8 KB mit RAM belegt sein, so loeschen Sie einfach Bit 1 des Prozessorports. Entsprechend setzen Sie Bit 1, wenn das KERNAL-ROM erreichbar sein soll.

```
53248-57343: RAM    : LORAM = 0 und HIRAM = 0
             I/O    : LORAM = 1 oder HIRAM = 1
```

Ist also im Bereich von 57344 bis 65535 KERNAL ausgewaehlt worden, so kann hier nicht RAM erreicht werden, da hierzu sowohl LORAM, als auch HIRAM geloescht sein muessen, was aber nicht moeglich ist, da ansonsten im obigen Bereich KERNAL nicht sichtbar waere. Im folgenden Bereich wird es etwas komplizierter:

```
40960-49151: RAM    : HIRAM = 0 oder
                      (HIRAM = 1 und LORAM = 0)
             BASIC  : LORAM = 1 und HIRAM = 1 und GAME = 1
             MODUL  : HIRAM = 1 und GAME = 0 und EXROM = 0
```

Auch hier ergibt sich zum Beispiel die Einschraenkung, dass kein BASIC verfuegbar sein kann, wenn nicht zusaetzlich im oberen Bereich KERNAL verfuegbar ist (umgekehrt ist dies jedoch durchaus moeglich und sinnvoll).

```
32768-40959: RAM    : HIRAM = 0 oder LORAM = 0 oder
                      (saemtliche vier Leitungen gesetzt)
             MODUL  : HIRAM = 1 und LORAM = 1 und EXROM = 0
```

Zusaetzlich existiert noch die Kombination "GAME = 0 und EXROM = 1" (restliche Leitungen bleiben unbeachtet), die verwendet wird, wenn Module fuer den ULTIMAX auf dem Commodore 64 ablaufen sollen. Von dieser Kombination wird hier jedoch abgesehen.

Anhand der obigen Angaben sollen nun die Leitungszustaende fuer eine gegebene Speicheraufteilung erarbeitet werden: KERNAL soll verfuegbar sein, ebenso der normale I/O-Bereich. Allerdings soll das gesamte Gebiet von 32768 bis 49151 durch ein im Cartridge vorhandenes ROM belegt sein. Die Vorgehensweise:

HIRAM muss gesetzt werden, da KERNAL verfuegbar sein soll. I/O ist automatisch auch verfuegbar, wenn HIRAM gesetzt ist. LORAM ist daher (bisher) beliebig setzbar. Zusaetzlich muessen nun auch noch GAME und EXROM geloescht sein, da HIRAM = 1, GAME = 0 und EXROM = 0 sein muessen, um im Bereich von 40960 bis 49151 ein Cartridge einzublenden. Um nun auch im darunterliegenden Bereich auf den Modulinhalt zugreifen zu koennen, muss also noch LORAM gesetzt werden. Die endgueltige Verteilung ist also ...

LORAM = 1, HIRAM = 1, GAME = 0, EXROM = 0

Genauso muss bei anderen Speicherverteilungen vorgegangen werden. Sollten notwendige Bedingungen fuer diese Aufteilungen sich widersprechen (zum Beispiel die Bedingung CARTRIDGE in einem Bereich und RAM im Bereich von 57344 bis 65535), so ist diese Kombination nicht moeglich und wohl auch in den meisten Faellen nicht sinnvoll.

Nun zur Bedeutung der Leitung CHAREN: Sie tauscht den Zeichengenerator (das sind uebrigens die vorhin noch als "fehlend" bezeichneten 4 KB der 20 KB ROM) gegen den I/O-Bereich aus. Soll der Zeichengenerator also ausgelesen werden, so muss dieses Bit geloescht werden. Die genaue Vorgehensweise ist bei der Definition eigener Zeichen (Video Interface Chip) dargelegt. Befindet sich momentan in diesem Bereich kein I/O, sondern RAM, so kann der Zeichengenerator NICHT ausgelesen werden, da dieser nur gegen I/O, nicht aber gegen RAM ausgetauscht werden kann.

Um nun mit dem gesamten RAM arbeiten zu koennen, muss also sowohl darauf schreibend, als auch lesend zugegriffen werden koennen. Schreibend ist dies, sofern das RAM mit ROM ueberlegt ist, durch POKE (oder durch die Aequivalente der Maschinensprache) moeglich. Nur das Lesen kann nicht auf diese Weise erfolgen. Es bietet sich also an, eine Art PEEK Befehl, der nur auf RAM wirkt, zu konstruieren.

Dies ist durch Verwendung der USR-Funktion problemlos moeglich: Zuerst das Maschinenprogramm (als Bereich wurde hier der Cassettenpuffer gewaehlt) mit den Erlaeuterungen (siehe auch ROM-Listing von 47117 bis 47139):

```
828 LDA     21        (20/21) fuer weitere Verwendungen auf
830 PHA               den Stack retten (POKE und WAIT)
831 LDA     20
833 PHA
834 JSR  47095        ) GETADR wertet Argument zu USR aus
837 LDA      1        Inhalt des Prozessorports
839 PHA               auf Stack legen
840 AND   #252        Bits 0 (LORAM) und 1 (HIRAM) loeschen
842 SEI               Interrupt verhindern (!!!)
843 STA      1        neue Speicherverteilung festsetzen
845 LDY     #0
847 LDA   (20),Y      Zeichen aus RAM lesen
849 TAY               und ins YR uebertragen
850 PLA               vorherigen Inhalt des
851 STA      1        Prozessorports wiederherstellen
853 CLI               Interrupts wieder zulassen
854 PLA
855 STA     20
857 PLA               Pointer in (20/21) wiederherstellen
858 STA     21
860 JMP  45986        ) Accu := 0, INTFLP
```

Nun kann durch USR jede Adresse der 64 KB RAM gelesen werden, und das von BASIC aus. So kann zum Beispiel eine hochaufloesende Graphik (bei deren Erstellung ja, wie zum Beispiel die Sinuskurve bei der Erklaerung des VIC-II-Chips, auch lesend auf den Bit-Map-Bereich zugegriffen werden muss) der Speicherbereich unter den Betriebssystem-ROMs verwendet werden, der ansonsten fuer BASIC brachliegt.

Und hier der zugehoerige BASIC-Loader:

```
100 DATA 165, 21, 72, 165, 20, 72, 32, 247, 183, 165, 1, 72
110 DATA 41, 252, 120, 133, 1, 160, , 177, 20, 168, 104, 133
120 DATA 1, 88, 104, 133, 20, 104, 133, 21, 76, 162, 179
130 FOR I = 828 TO 862 : READ A : POKE I, A : NEXT
140 POKE 785, 60 : POKE 786, 3
```

Um die Sinuskurve aus der Erklaerung des Video-Chips nun im Bereich der hinteren acht Kilobytes abzulegen, muss zuerst obiges Programm eingegeben und gestartet werden. Danach kann es durch "NEW" geloescht werden, da sich das Maschinenprogramm im Bereich von 828 bis 862 befindet. Geben Sie dann folgendes Programm ein:

```
200 POKE 56576, PEEK (56576) AND 252 : POKE 53280, 14
210 FOR I = 57344 TO 65535 : POKE I, 0 : NEXT
220 FOR I = 50176 TO 51175 : POKE I, 1 : NEXT
230 POKE 53272, PEEK (53272) OR 8
240 POKE 53265, PEEK (53265) OR 32
250 FOR X = 0 TO 319 : Y = 100 - 100 * SIN (X * pi / 160)
260 B = 57344 + (X AND 504) + 40 * (Y AND 248) + (Y AND 7)
270 POKE B, USR (B) OR 2 ↑ (7 - (X AND 7)) : NEXT
280 GET G$ : IF G$ = "" GOTO 280
290 POKE 56576, PEEK (56576) OR 3 : PRINT CHR$ (147)
300 POKE 53272, PEEK (53272) AND 247
```

PINBELEGUNG DER CPU 6510:

Pin	Belegung
Pin 1	PHO IN, Eingang des Systemtakts (um ein MHz)
Pin 2	RDY, Ready-Leitung (dient im Zusammenhang mit dem Anhalten des Prozessor, liegt RDY auf eins, so stoppt der Prozessor im naechsten Lesezyklus)
Pin 3	IRQ/, Interrupt-Eingang des Prozessors, dient zum Ausloesen eines Interrupts zum Beispiel durch die CIAs oder den VIC-II-Chip
Pin 4	NMI/, Eingang zum Ausloesen des nichtmaskierbaren Interrupts
Pin 5	AEC/, Address Enable Control, bringt Prozessorbus in hochohmigen Zustand und laesst dadurch den Zugriff anderer Bausteine auf den Bus zu, zum Beispiel durch den VIC-II-Chip)
Pin 6	VCC, Versorgungsspannung von +5 Volt
Pin 7 - 20	A0 bis A13, Adressbus des Prozessors
Pin 21	GND, Masseleitung
Pin 22 - 23	A14 und A15, Adressbus des Prozessors
Pin 24 - 29	P5 bis P0, Prozessorportleitungen
Pin 30 - 37	DB7 bis DB0, Datenbus des Prozessors
Pin 38	R/W, Schreib/Leseleitung zum Zugriff auf Bus
Pin 39	PH2 OUT, zweiter Systemtakt zur Steuerung anderer Bausteine
Pin 40	RES/, Reset-Leitung dient zur Startinitialisierung des Systems, bei Uebergang von 0 (Ruhezustand) nach 1 wird RESET-Vorgang begonnen.

PIN CONFIGURATION

MOS 6510

Signal	Pin	Pin	Signal
PH0 IN	1	40	$\overline{RES}$
RDY	2	39	PH2 OUT
$\overline{IRQ}$	3	38	R/$\overline{W}$
$\overline{NMI}$	4	37	DB0
$\overline{AEC}$	5	36	DB1
VCC	6	35	DB2
A0	7	34	DB3
A1	8	33	DB4
A2	9	32	DB5
A3	10	31	DB6
A4	11	30	DB7
A5	12	29	P0
A6	13	28	P1
A7	14	27	P2
A8	15	26	P3
A9	16	25	P4
A10	17	24	P5
A11	18	23	A15
A12	19	22	A14
A13	20	21	GND

```
40960 148 227   58260   Sprungvektor fuer BASIC-Kaltstart (BASIC-RESET)
40962 123 227   58235   Sprungvektor fuer BASIC-Warmstart, wird bei NMI
                        verwendet (CLRCHN, Tastatur aktivieren, Descrip-
                        torindex und Stack ruecksetzen, 'CONT' sperren,
                        "READY.", Eingabewarteschleife)

40964  67  66  77  66  65  83  73  67   "CBMBASIC"
```

Vektortabelle der BASIC-Befehle

Die folgenden Adressen bis einschliesslich 'NEW' werden ab 43001 auf den Stack gelegt und durch das RTS der CHRGET aufgerufen. Die Routinen beginnen daher jeweils an der naechsthoeheren Adresse.

```
40972  48 168   43056   128 END
40974  65 167   42817   129 FOR
40976  29 173   44317   130 NEXT
40978 247 168   43255   131 DATA
40980 164 171   43940   132 INPUT#
40982 190 171   43966   133 INPUT
40984 128 176   45184   134 DIM
40986   5 172   44037   135 READ
40988 164 169   43428   136 LET
40990 159 168   43167   137 GOTO
40992 112 168   43120   138 RUN
40994  39 169   43303   139 IF
40996  28 168   43036   140 RESTORE
40998 130 168   43138   141 GOSUB
41000 209 168   43217   142 RETURN
41002  58 169   43322   143 REM
41004  46 168   43054   144 STOP
41006  74 169   43338   145 ON
41008  44 184   47148   146 WAIT
41010 103 225   57703   147 LOAD
41012  85 225   57685   148 SAVE
41014 100 225   57700   149 VERIFY
41016 178 179   46002   150 DEF
41018  35 184   47139   151 POKE
41020 127 170   43647   152 PRINT#
41022 159 170   43679   153 PRINT
41024  86 168   43094   154 CONT
41026 155 166   42651   155 LIST
41028  93 166   42589   156 CLR
41030 133 170   43653   157 CMP
41032  41 225   57641   158 SYS
41034 189 225   57789   159 OPEN
41036 198 225   57798   160 CLOSE
41038 122 171   43898   161 GET
41040  65 166   42561   162 NEW
```

Vektortabelle der BASIC-Funktionen

Die Adressen der Funktionen werden ab 45014 nach (85/86) gebracht und dann durch JSR 84 (in 84 steht der Code fuer JMP) aufgerufen.

```
41042  57 188   48185   180 SGN
41044 204 188   48332   181 INT
41046  88 188   48216   182 ABS
41048  16   3     784   183 USR
41050 125 179   45949   184 FRE
41052 158 179   45982   185 POS
41054 113 191   49009   186 SQR
41056 151 224   57495   187 RND
41058 234 185   47594   188 LOG
41060 237 191   49133   189 EXP
41062 100 226   57956   190 COS
41064 107 226   57963   191 SIN
41066 180 226   58036   192 TAN
41068  14 227   58126   193 ATN
41070  13 184   47117   194 PEEK
41072 124 183   46972   195 LEN
41074 101 180   46181   196 STR$
41076 173 183   47021   197 VAL
41078 139 183   46987   198 ASC
41080 236 182   46828   199 CHR$
41082   0 183   46848   200 LEFT$
41084  44 183   46892   201 RIGHT$
41086  55 183   46903   202 MID$
```

Tabelle der Prioritaetsflags und Adressen fuer die Formelauswertung

Die folgende Tabelle enthaelt fuer die angegebenen Funktionen jeweils ein Prioritaetsflag und die Startadresse. Die Adresse wird ab 44576 auf den Stack abgelegt und durch RTS aufgerufen. Die Routinen beginnen daher also jeweils an der naechsthoeheren Adresse.

```
41088 121 105 184   121 47209   170 +
41091 121  82 184   121 47186   171 -
41094 123  42 186   123 47658   172 *
41097 123  17 187   123 47889   173 /
41100 127 122 191   127 49018   174 ↑
41103  80 232 175    80 45032   175 AND
41106  70 229 175    70 45029   176 OR
41109 125 179 191   125 49075       Vorzeichenwechsel
41112  90 211 174    90 44755   168 NOT
41115 100  21 176   100 45077   177 >    178 =    179 <
```

Tabelle der BASIC-Befehlsworte fuer Tokenumwandlung und LIST

```
41118  69  78 196                                  END
41121  70  79 210                                  FOR
41124  78  69  88 212                              NEXT
41128  68  65  84 193                              DATA
41132  73  78  80  85  84 163                      INPUT#
41138  73  78  80  85 212                          INPUT
41143  68  73 205                                  DIM
41146  82  69  65 196                              READ
41150  76  69 212                                  LET
41153  71  79  84 207                              GOTO
41157  82  85 206                                  RUN
41160  73 198                                      IF
41162  82  69  83  84  79  82 197                  RESTORE
41169  71  79  83  85 194                          GOSUB
41174  82  69  84  85  82 206                      RETURN
41180  82  69 205                                  REM
41183  83  84  79 208                              STOP
41187  79 206                                      ON
41189  87  65  73 212                              WAIT
41193  76  79  65 196                              LOAD
41197  83  65  86 197                              SAVE
41201  86  69  82  73  70 217                      VERIFY
41207  68  69 198                                  DEF
41210  80  79  75 197                              POKE
41214  80  82  73  78  84 163                      PRINT#
41220  80  82  73  78 212                          PRINT
41225  67  79  78 212                              CONT
41229  76  73  83 212                              LIST
41233  67  76 210                                  CLR
41236  67  77 196                                  CMD
41239  83  89 211                                  SYS
41242  79  80  69 206                              OPEN
41246  67  76  79  83 197                          CLOSE
41251  71  69 212                                  GET
41254  78  69 215                                  NEW
```

Tabelle der BASIC-Befehlsworte fuer Tokenumwandlung und LIST (Fortsetzung)

```
41257  84  65  66 168                    TAB(
41261  84 207                            TO
41263  70 206                            FN
41265  83  80  67 168                    SPC(
41269  84  72  69 206                    THEN
41273  78  79 212                        NOT
41276  83  84  69 208                    STEP
41280 171                                +
41281 173                                -
41282 170                                *
41283 175                                /
41284 222                                ↑
41285  65  78 196                        AND
41288  79 210                            OR
41290 190                                >
41291 189                                =
41292 188                                <
41293  83  71 206                        SGN
41296  73  78 212                        INT
41299  65  66 211                        ABS
41302  85  83 210                        USR
41305  70  82 197                        FRE
41308  80  79 211                        POS
41311  83  81 210                        SQR
41314  82  78 196                        RND
41317  76  79 199                        LOG
41320  69  88 208                        EXP
41323  67  79 211                        COS
41326  83  73 206                        SIN
41329  84  65 206                        TAN
41332  65  84 206                        ATN
41335  80  69  69 203                    PEEK
41339  76  69 206                        LEN
41342  83  84  82 164                    STR$
41346  86  65 204                        VAL
41349  65  83 195                        ASC
41352  67  72  82 164                    CHR$
41356  76  69  70  84 164                LEFT$
41361  82  73  71  72  84 164            RIGHT$
41367  77  73  68 164                    MID$
41371  71 207                            GO
41373   0                                Trenncode
```

BASIC-Fehlermeldungen

41374	84	79	79	32	77	65	78	89	32	70	73	76	69	211	TOO MANY FILES
41388	70	73	76	69	32	79	80	69	206						FILE OPEN
41397	70	73	76	69	32	78	79	84	32	79	80	69	206		FILE NOT OPEN
41410	70	73	76	69	32	78	79	84	32	70	79	85	78	196	FILE NOT FOUND
41424	68	69	86	73	67	69	32	78	79	84	32				DEVICE NOT
41435	80	82	69	83	69	78	212								PRESENT
41442	78	79	84	32	73	78	80	85	84	32	70	73	76	197	NOT INPUT FILE
41456	78	79	84	32	79	85	84	80	85	84	32				NOT OUTPUT
41467	70	73	76	197											FILE
41471	77	73	83	83	73	78	71	32	70	73	76	69	32		MISSING FILE
41484	78	65	77	197											NAME
41488	73	76	76	69	71	65	76	32							ILLEGAL
41496	68	69	86	73	67	69	32	78	85	77	66	69	210		DEVICE NUMBER
41509	78	69	88	84	32	87	73	84	72	79	85	84	32		NEXT WITHOUT
41522	70	79	210												FOR
41525	83	89	78	84	65	216									SYNTAX
41531	82	69	84	85	82	78	32								RETURN
41538	87	73	84	72	79	85	84	32	71	79	83	85	194		WITHOUT GOSUB
41551	79	85	84	32	79	70	32	68	65	84	193				OUT OF DATA
41562	73	76	76	69	71	65	76	32							ILLEGAL
41562	81	85	65	78	84	73	84	217							QUANTITY
41578	79	86	69	82	70	76	79	215							OVERFLOW
41586	79	85	84	32	79	70	32	77	69	77	79	82	217		OUT OF MEMORY
41599	85	78	68	69	70	39	68	32							UNDEF'D
41607	83	84	65	84	69	77	69	78	212						STATEMENT
41616	66	65	68	32	83	85	66	83	67	82	73	80	212		BAD SUBSCRIPT
41629	82	69	68	73	77	39	68	32	65	82	82	65	217		REDIM'D ARRAY
41642	68	73	86	73	83	73	79	78	32	66	89	32			DIVISION BY
41654	90	69	82	207											ZERO
41658	73	76	76	69	71	65	76	32	68	73	82	69	67	212	ILLEGAL DIRECT
41672	84	89	80	69	32	77	73	83	77	65	84	67	200		TYPE MISMATCH
41685	83	84	82	73	78	71	32	84	79	79	32				STRING TOO
41696	76	79	78	199											LONG
41700	70	73	76	69	32	68	65	84	193						FILE DATA
41709	70	79	82	77	85	76	65	32	84	79	79	32			FORMULA TOO
41721	67	79	77	80	76	69	216								COMPLEX
41728	67	65	78	39	84	32	67	79	78	84	73	78	85	197	CAN'T CONTINUE
41742	85	78	68	69	70	39	68	32							UNDEF'D
41750	70	85	78	67	84	73	79	206							FUNCTION
41758	86	69	82	73	70	217									VERIFY
41764	76	79	65	196											LOAD

Startadressen der BASIC-Fehlermeldungen

```
41768 158 161   41374    1 - TOO MANY FILES
41770 172 161   41388    2 - FILE OPEN
41772 181 161   41397    3 - FILE NOT OPEN
41774 194 161   41410    4 - FILE NOT FOUND
41776 208 161   41424    5 - DEVICE NOT PRESENT
41778 226 161   41442    6 - NOT INPUT FILE
41780 240 161   41456    7 - NOT OUTPUT FILE
41782 255 161   41471    8 - MISSING FILE NAME
41784  16 162   41488    9 - ILLEGAL DEVICE NUMBER
41786  37 162   41509   10 - NEXT WITHOUT FOR
41788  53 162   41525   11 - SYNTAX
41790  59 162   41531   12 - RETURN WITHOUT GOSUB
41792  79 162   41551   13 - OUT OF DATA
41794  90 162   41562   14 - ILLEGAL QUANTITY
41796 106 162   41578   15 - OVERFLOW
41798 114 162   41586   16 - OUT OF MEMORY
41800 127 162   41599   17 - UNDEF'D STATEMENT
41802 144 162   41616   18 - BAD SUBSCRIPT
41804 157 162   41629   19 - REDIM'D ARRAY
41806 170 162   41642   20 - DIVISION BY ZERO
41808 186 162   41658   21 - ILLEGAL DIRECT
41810 200 162   41672   22 - TYPE MISMATCH
41812 213 162   41685   23 - STRING TOO LONG
41814 228 162   41700   24 - FILE DATA
41816 237 162   41709   25 - FORMULA TOO COMPLEX
41818   0 163   41728   26 - CAN'T CONTINUE
41820  14 163   41742   27 - UNDEF'D FUNCTION
41822  30 163   41758   28 - VERIFY
41824  36 163   41764   29 - LOAD
41826 131 163   41859   30 - BREAK
```

weitere Meldungen

```
41828  13  79  75  13   0                               OK
41833  32  32  69  82  82  79  82   0                   ERROR
41841  32  73  78  32   0                               IN
41846  13  10  82  69  65  68  89  46  13  10   0       READY.
41857  13  10  66  82  69  65  75   0                   BREAK
41865 160                                               ?
```

Stack-Suchroutine fuer 'FOR', 'NEXT' und 'RETURN'

```
41866 TSX                Bei den naechsten vier Bytes im Stack handelt
41867 INX                es sich um zwei Ruecksprungadressen (Ruecksprung
41868 INX                in Interpreterschleife und Ruecksprung zur
41869 INX                aufrufenden Routine)
41870 INX
41871 LDA    257,X       Ist naechstes Stackbyte identisch mit
41874 CMP   #129         'FOR'-Code?
41876 BNE  41911         Nein: Schleife nicht gefunden, RTS
41878 LDA     74         Ist FOR-NEXT-Variablenpointer high = 0? (vgl. 44320)
41880 BNE  41892         (keine Variablenangabe bei NEXT) Nein: weiter bei 41892
41882 LDA    258,X       Variablenpointer der innersten Schleife
41885 STA     73         aus dem Stack nach (73/74) bringen
41887 LDA    259,X
41890 STA     74
41892 CMP    259,X       (73/74) mit Variablenpointer im Stack vergleichen
41895 BNE  41904         Ungleich: naechste Schleife suchen
41897 LDA     73
41899 CMP    258,X
41902 BEQ  41911         Gleich: Schleife gefunden, RTS
41904 TXA
41905 CLC                Suchzeiger um 18
41906 ADC    #18         (Anzahl Bytes je FOR-NEXT-Schleife) erhoehen
41908 TAX
41909 BNE  41871         Unbedingter Sprung zum Scheifenbeginn
41911 RTS
```

Aufbau der Datensaetze von FOR-NEXT und GOSUB im Stack

```
-----------------------------------------------------------------
|              FOR-NEXT              |            GOSUB           |
-----------------------------------------------------------------
| 0 Schleifenstartadresse low  | 0 CHRGET-Pointer high       |
| 1                       high | 1                low        |
-----------------------------------------------------------------
| 2 Zeilennummer          high | 2 Zeilennummer   high       |
| 3                       low  | 3                low        |
-----------------------------------------------------------------
| 4 TO-Wert     LSB            | 4 GOSUB-Code 141            |
| 5                            |------------------------------
| 6                            |
| 7             MSB, Vorzeichen|
| 8             Exponent       |
--------------------------------
| 9 STEP-Wert   Vorzeichen     |
|10             LSB            |
|11                            |
|12                            |
|13             MSB            |
|14             Exponent       |
--------------------------------
|15 Variablenpointer      high |
|16                       low  |
--------------------------------
|17 FOR-Code 129               |
--------------------------------
```

Block-Verschiebe-Routine

Einbau von Programmzeilen und Verschieben von Arrays

```
Eingabe: Quellbereich: Startadresse: (95/96)
                       Endadresse+1: (90/91)
         Zielbereich:  Endadresse+1: (88/89) sowie (Accu/YR)

41912 JSR  41992       > Pruefung, ob genug freier Platz im Speicher
41915 STA     49       (Accu/YR) als neue Startadresse
41917 STY     50       fuer freien Speicherplatz speichern
41919 SEC
41920 LDA     90       Startadresse von der Endadresse des
41922 SBC     95       Quellbereich subtrahieren
41924 STA     34       ergibt Laenge des Quellblock low
41926 TAY
41927 LDA     91       Ebenso Bytes high subtrahieren,
41929 SBC     96
41931 TAX              ergibt Zaehler der vollen 256-Byte-Abschnitte
41932 INX
41933 TYA              Ist ein Restabschnitt vorhanden?
41934 BEQ  41971       Nein: nur ganze Pages verschieben
41936 LDA     90       Endadresse+1 des Quellbereichs
41938 SEC
41939 SBC     34       minus Laenge des Restabschnitts
41941 STA     90       ergibt Adresse des Restabschnitts
41943 BCS  41948
41945 DEC     91
41947 SEC
41948 LDA     88       Endadresse+1 des Zielbereichs
41950 SBC     34       minus Laenge des Restabschnitts
41952 STA     88       ergibt Adresse des neuen Restabschnitts
41954 BCS  41964
41956 DEC     89
41958 BCC  41964       Unbedingter Sprung

41960 LDA    (90),Y    Transfer-Schleife fuer Restabschnitt
41962 STA    (88),Y
41964 DEY
41965 BNE  41960
41967 LDA    (90),Y    Transfer-Schleife fuer ganze Pages
41969 STA    (88),Y
41971 DEC     91
41973 DEC     89
41975 DEX              Zaehler fuer ganze Pages vermindern
41976 BNE  41964
41978 RTS
```

Test, ob genug Platz im Stack (Anzahl benoetigter Unterprogrammebenen im Accu)

```
41979 ASL              Accu := 2 * Accu
41980 ADC    #62       + 62
41982 BCS  42037       Ist Accu beim Aufruf > 96
41984 STA     34
41986 TSX              oder ist der Stackpointer
41987 CPX     34       < 2 * (Accu) + 62?
41989 BCC  42037       Ja: "OUT OF MEMORY ERROR"
41991 RTS
```

Test, ob genug Platz im Arbeitsspeicher

```
41992 CPY     52       (Accu/YR) ist Endadresse, bis zu
41994 BCC  42036       der Speicherplatz benoetigt wird
41996 BNE  42002
41998 CMP     51       Ist diese Adresse kleiner als momentaner
42000 BCC  42036       String-Anfangszeiger? Ja: RTS
42002 PHA              Accu, YR und (87,...,96) auf
42003 LDX     #9       Stack zwischenspeichern
42005 TYA
42006 PHA
42007 LDA     87,X
42009 DEX
42010 BPL  42006
42012 JSR  46374       > GARBAGE COLLECT, Stringmuellbeseitigung
42015 LDX   #247       Accu, YR und (87,...,96) wiederherstellen
42017 PLA
42018 STA     97,X
42020 INX
42021 BMI  42017
42023 PLA
42024 TAY
42025 PLA
42026 CPY     52       Nochmals (Accu/YR) mit (51/52) vergleichen
42028 BCC  42036       Jetzt genug Platz? Ja: RTS
42030 BNE  42037
42032 CMP     51
42034 BCS  42037       sonst "OUT OF MEMORY ERROR"
42036 RTS
```

Ausgabe von Fehlermeldungen und READY-Status

```
42037 LDX    #16       Code fuer "OUT OF MEMORY"
42039 JMP   (768)      Normalwert des Vektors (768/769): 58251
42042 TXA
42043 ASL              Accu := 2 * Accu
42044 TAX
42045 LDA  41766,X     Startadresse der Fehlermeldung
42048 STA     34       aus Tabelle in Pointer (34/35) bringen
42050 LDA  41767,X
42053 STA     35
42055 JSR  65484       > CLRCHN, alle Kanaele schliessen
42058 LDA     #0
42060 STA     19       aktiver Eingabe-Kanal := Tastatur
42062 JSR  43735       > Zeilenvorschub drucken
42065 JSR  43845       > "?" drucken
42068 LDY     #0
42070 LDA    (34),Y    Fehlermeldung zeichenweise in Accu laden
42072 PHA
42073 AND   #127       Bit 7 loeschen (ist beim letzen Zeichen = 1)
42075 JSR  43847       > Zeichen im Accu ausgeben
42078 INY
42079 PLA              War im letzen Zeichen Bit 7 gesetzt?
42080 BPL  42070       Nein: weiterdrucken ...
42082 JSR  42618       > CONT sperren, Stack und Descriptorindex ruecksetzen
42085 LDA   #105       (Accu/YR) := 41833, Startadresse von " ERROR"
42087 LDY   #163
42089 JSR  43806       > " ERROR" drucken
42092 LDY     58       Zeilennummer high (ist im Direktmodus = 255)
42094 INY
42095 BEQ  42100       Direktomodus? Ja: weiter bei 42100
42097 JSR  48578       > " IN " und Zeilennummer drucken
```

Ausgabe von "READY.", Eingabewarteschleife

```
42100 LDA   #118      (Accu/YR) := 41846, Startadresse von "READY."
42102 LDY   #163
42104 JSR  43806      > "READY." drucken
42107 LDA   #128      Direktmodus einschalten
42109 JSR  65424      > SETMSG, Ausgabemodus setzen

42112 JMP   (770)     Normalwert des Vektors (770/771): 42115
42115 JSR  42336      > Eingabe-Warteschleife, zurueck nach 'RETURN'
42118 STX    122      CHRGET-Pointer auf 511 (ein Byte vor Eingabepuffer)
42120 STY    123      setzen
42122 JSR    115      > CHRGET holt naechstes Zeichen
42125 TAX             setzt Zero-Flag, falls Accu = 0
42126 BEQ  42112      keine Eingabe, zurueck zur Warteschleife
42128 LDX   #255
42130 STX     58      Flag fuer Direktmodus setzen
42132 BCC  42140      Ist erstes Zeichen eine Ziffer? Ja: Programmzeile
42134 JSR  42361      > Text im Eingabepuffer in BASIC-Tokens umwandeln
42137 JMP  42977      > Interpreterschleife, eingegebene Zeile ausfuehren
```

Loeschen und Einfuegen von Programmzeilen

```
42140 JSR  43371      > Zeilennummer nach (20/21) bringen
42143 JSR  42361      > Text im Eingabepuffer in BASIC-Tokens umwandeln
42146 STY     11      Laenge der Zeile
42148 JSR  42515      > Startadresse der Zeile nach (95/96)
42151 BCC  42221      Zeile bereits vorhanden? Nein: kein Loeschen
42153 LDY     #1
42155 LDA    (95),Y   Startadresse high der naechsten Zeile
42157 STA     35
42159 LDA     45      Variablen-Anfangspointer low
42161 STA     34
42163 LDA     96      Startadresse high der zu loeschenden Zeile
42165 STA     37
42167 LDA     95      Startadresse low der zu loeschenden Zeile
42169 DEY
42170 SBC    (95),Y   Startadresse low der naechsten Zeile
42172 CLC
42173 ADC     45      Variablen-Anfangspointer low
42175 STA     45      ergibt neuen Variablen-Anfangspointer low
42177 STA     36
42179 LDA     46      ebenso high
42181 ADC   #255
42183 STA     46
42185 SBC     96      Startadresse high der zu loeschenden Zeile
42187 TAX             ergibt Anzahl der zu verschiebenden Pages
42188 SEC
42189 LDA     95      Startadresse low der zu loeschenden Zeile
42191 SBC     45      Variablen-Anfangspointer low
42193 TAY             Laenge des Restabschnitts
42194 BCS  42199      > 255? Nein: weiter bei 42199
42196 INX             Zaehler fuer 256-Byte-Bloecke um eins erhoehen
42197 DEC     37      Transportzeiger initialisieren
42199 CLC
42200 ADC     34
42202 BCC  42207
42204 DEC     35
42206 CLC
42207 LDA    (34),Y   Transfer-Schleife
42209 STA    (36),Y
42211 INY
42212 BNE  42207
```

Loeschen und Einfuegen von Programmzeilen (Fortsetzung)

```
42214 INC     35
42216 INC     37
42218 DEX
42219 BNE  42207
42221 JSR  42585       > CHRGET-Pointer ruecksetzen, CLR
42224 JSR  42291       > Linkpointer fuer Zeilen neu berechnen
42227 LDA    512       War erstes Zeichen im Eingabepuffer = 0?
42230 BEQ  42112       Ja: Zeile nur loeschen, zurueck zur Warteschleife

42232 CLC
42233 LDA     45       Variablenanfangszeiger low
42235 STA     90       Endadresse+1 (Quellbereich) low
42237 ADC     11       Laenge der Zeile
42239 STA     88       Endadresse+1 (Zielbereich) low
42241 LDY     46       Variablenanfangszeiger high
42243 STY     91       Endadresse+1 (Quellbereich) high
42245 BCC  42248
42247 INY              Uebertrag addieren
42248 STY     89       Endadresse+1 (Zielbereich) high
42250 JSR  41912       > Block-Verschiebe-Routine
42253 LDA     20       Zeilennummer in (20/21)
42255 LDY     21
42257 STA    510       vor Eingabepuffer setzen
42260 STY    511
42263 LDA     49       neuer Variablen-Endezeiger (vgl. 41915)
42265 LDY     50
42267 STA     45       Pointer auf Programmende+1
42269 STY     46
42271 LDY     11       Zeilenlaenge
42273 DEY
42274 LDA    508,Y     Verschiebeschleife transferiert die neue Zeile aus
42277 STA    (95),Y    Eingabepuffer in den Arbeitsspeicher
42279 DEY
42280 BPL  42274
42282 JSR  42585       > CHRGET-Pointer ruecksetzen, CLR
42285 JSR  42291       > Linkpointer fuer Zeilen neu berechnen
42288 JMP  42112       > Zurueck zur Eingabe-Warteschleife
```

Linkpointer fuer Zeilen neu berechnen

```
42291 LDA     43       Programm-Anfangspointer
42293 LDY     44
42295 STA     34       als Suchzeiger nach (34/35) speichern
42297 STY     35
42299 CLC
42300 LDY     #1
42302 LDA    (34),Y    Linkadresse high
42304 BEQ  42335       = 0? Ja: Programmende erreicht, RTS
42306 LDY     #4       YR auf erstes Zeichen des Programmtextes setzen
42308 INY
42309 LDA    (34),Y    Naechstes Zeichen = 0?
42311 BNE  42308       Nein: Zeile noch nicht zu Ende, weitersuchen
```

Linkpointer fuer Zeilen neu berechnen (Fortsetzung)

```
42313 INY
42314 TYA                Zeilenlaenge
42315 ADC      34        Pointer low auf aktuelle Zeile
42317 TAX
42318 LDY      #0
42320 STA    (34),Y      als Linkpointer low speichern
42322 LDA      35        Pointer high auf aktuelle Zeile
42324 ADC      #0        Uebertrag addieren
42326 INY
42327 STA    (34),Y      als Linkpointer high speichern
42329 STX      34        Startadresse der naechsten Zeile
42331 STA      35        nach (34/35) speichern
42333 BCC   42300        Unbedingter Sprung zum Zeilenanfang
42335 RTS
```

Eingabe-Warteschleife, zurueck nach 'RETURN'

```
42336 LDX      #0        Zeiger in Eingabepuffer
42338 JSR   57618        > Zeichen von Tastatur in Accu
42341 CMP     #13        Carriage Return?
42343 BEQ   42358        Ja: Abschluss
42345 STA     512,X      Zeichen im Puffer speichern
42348 INX                Zeiger erhoehen
42349 CPX     #89        Puffer voll?
42351 BCC   42338        Nein: weitermachen ...
42353 LDX     #23        Code fuer "STRING TOO LONG"
42355 JMP   42039        > Fehlermeldung, READY.
42358 JMP   43722        > Abschluss der Eingabe
```

Text im Eingabepuffer in BASIC-Tokens umwandeln

```
42361 JMP    (772)       Normalwert des Vektors (772/773): 42364
42364 LDX     122        (122/123) zeigt auf erstes Zeichen im Eingabepuffer
42366 LDY      #4        Zeiger fuer codierte Zeile
42368 STY      15        'DATA'-Flag loeschen
42370 LDA     512,X      Zeichen aus Eingabepuffer holen
42373 BPL   42382        ASCII < 128 (ungeshiftet)? Ja: weiter bei 42382
42375 CMP    #255        Code fuer PI?
42377 BEQ   42441        Ja: abspeichern
42379 INX                Zeichen ueberlesen
42380 BNE   42370        Unbedingter Sprung zum Schleifenanfang

42382 CMP     #32        Leercode?
42384 BEQ   42441        Ja: abspeichern
42386 STA       8        Code nach (8) speichern (wenn Code = 34 vgl. 42474)
42388 CMP     #34        Anfuehrungszeichen?
42390 BEQ   42478        Ja: weiter bei 42478
42392 BIT      15        Bit 6 gesetzt (DATA)?
42394 BVS   42441        Ja: ASCII ungeaendert abspeichern
42396 CMP     #63        Code fuer Fragezeichen (Abkuerzung fuer 'PRINT')?
42398 BNE   42404        Nein: weiter bei 42404
42400 LDA    #153        Token fuer 'PRINT' abspeichern
42402 BNE   42441        Unbedingter Sprung
```

Text im Eingabepuffer in BASIC-Tokens umwandeln (Fortsetzung)

```
42404 CMP   #48       Code < 48?
42406 BCC  42412      Ja: weiter bei 42412
42408 CMP   #60       Code > 60?
42410 BCC  42441      Nein: weiter bei 42441
42412 STY   113       Pointer auf codierte Zeile merken
42414 LDY    #0
42416 STY    11       Wortzaehler fuer Tokentabelle initialisieren
42418 DEY
42419 STX   122       Zeiger auf Eingabepuffer zwischenspeichern
42421 DEX             (falls Wort nicht gefunden)
42422 INY
42423 INX
42424 LDA   512,X     Vom Zeichencode im Puffer
42427 SEC
42428 SBC  41118,Y    Zeichencode aus der Befehlstabelle subtrahieren
42431 BEQ  42422      Ergebnis = 0? Ja: naechstes Zeichen ...
42433 CMP  #128       Ergebnis = 128 (letztes Zeichen im Wort oder Abkuerzung)?
42435 BNE  42485      Nein: Befehl nicht gefunden, weiter bei 42485
42437 ORA    11       Bit 7 im Wortzaehler setzen, ergibt Code fuer Token
42439 LDY   113       Pointer auf codierte Zeile wieder holen
42441 INX
42442 INY
42443 STA   507,Y     Code abspeichern
42446 LDA   507,Y     Zero-Flag setzen
42449 BEQ  42505      War Code = 0 (Endezeichen)? Ja: weiter bei 42505
42451 SEC
42452 SBC   #58       Code fuer Trennungszeichen ":"?
42454 BEQ  42460      Ja: weiter bei 42460
42456 CMP   #73       'DATA'-Code? (131 - 58 = 73)
42458 BNE  42462      Nein: weiter bei 42462
42460 STA    15       Flag fuer 'DATA' setzen
42462 SEC
42463 SBC   #85       Code fuer 'REM'? (143 - 58 = 85)
42465 BNE  42370      Nein: zurueck zum Schleifenanfang
42467 STA     8       (8) := 0, falls Code fuer 'REM'
42469 LDA   512,X     naechstes Zeichen aus Puffer
42472 BEQ  42441      Endezeichen? Ja: weiter bei 42441
42474 CMP     8       weiterhin Abspeicherung als ASCII? (vgl. 42386)
42476 BEQ  42441      Nein: weiter bei 42441
42478 INY
42479 STA   507,Y     Code abspeichern
42482 INX
42483 BNE  42469      Sprung zum Schleifenanfang (ASCII Abspeicherung)

42485 LDX   122       Zeiger auf Eingabepuffer wiederherstellen
42487 INC    11       Wortzaehler erhoehen
42489 INY
42490 LDA  41117,Y    Zeiger auf Anfang des naechsten Befehls setzen
42493 BPL  42489
42495 LDA  41118,Y    Tabelle zu Ende?
42498 BNE  42424      Nein: weiter bei 42424
42500 LDA   512,X     Naechstes Zeichen aus Eingabepuffer
42503 BPL  42439      < 128? Ja: weiter bei 42439
42505 STA   509,Y     Code im Puffer speichern
42508 DEC   123       CHRGET-Pointer auf 511 ruecksetzen
42510 LDA  #255
42512 STA   122
42514 RTS
```

Berechnung der Startadresse einer Programmzeile

Eingabe: Zeilennummer in (20/21)

Ausgabe: Zeile vorhanden:
Carry = 1, Startadresse der Zeile in (95/96)
Zeile nicht vorhanden:
Carry = 0, Startadresse der naechsten Zeile in (95/96)

```
42515 LDA     43       Pointer auf Programmstartadresse
42517 LDX     44
42519 LDY     #1       Index fuer Linkpointer high
42521 STA     95
42523 STX     96
42525 LDA    (95),Y    Linkpointer high
42527 BEQ  42560       = 0 (Programmende)? Ja: Zeile nicht gefunden
42529 INY
42530 INY              YR := 3
42531 LDA     21       gesuchte Zeilennummer high mit
42533 CMP    (95),Y    Zeilennummer der aktuellen Programmzeile vergleichen
42535 BCC  42561       kleiner? Ja: Zeile nicht gefunden
42537 BEQ  42542       gleich? Ja: weiter bei 42542
42539 DEY              YR := 2
42540 BNE  42551       Unbedingter Sprung

42542 LDA     20       gesuchte Zeilennummer low mit
42544 DEY              YR := 2
42545 CMP    (95),Y    Zeilennummer der aktuellen Programmzeile vergleichen
42547 BCC  42561       kleiner? Ja: Zeile nicht gefunden
42549 BEQ  42561       gleich? Ja: Ruecksprung mit Carry = 1
42551 DEY              YR := 1
42552 LDA    (95),Y    Linkpointer high in XR
42554 TAX
42555 DEY              YR := 0
42556 LDA    (95),Y    Linkpointer low in Accu
42558 BCS  42519       Unbedingter Sprung zum Schleifenanfang

42560 CLC              Flag fuer 'Zeile nicht vorhanden' setzen
42561 RTS
```

BASIC-Routine NEW (enthaelt CLR)

```
42562 BNE  42561       Folgt Trennungszeichen? Nein: "SYNTAX ERROR"
42564 LDA     #0
42566 TAY
42567 STA    (43),Y    Null in die ersten beiden Stellen des Programmspeichers
42569 INY              (2049, 2050) schreiben
42570 STA    (43),Y
42572 LDA     43       Programm-Anfangpointer low
42574 CLC
42575 ADC     #2       + 2
42577 STA     45       ergibt Variablen-Anfangpointer low,
42579 LDA     44       ebenso Programm-Anfangspointer high
42581 ADC     #0       Uebertrag addieren
42583 STA     46       ergibt Variablen-Anfangspointer high
42585 JSR  42638       > CHRGET-Pointer ruecksetzen
42588 LDA     #0       Setzen der Zero-Flag fuer Uebergang zu CLR
```

BASIC-Routine CLR

```
42590 BNE   42637       Folgt Trennungszeichen? Nein: "SYNTAX ERROR"
42592 JSR   65511       > CLALL, alle Kanaele loeschen
42595 LDA      55       oberste RAM-Grenze
42597 LDY      56
42599 STA      51       String Anfangspointer
42601 STY      52
42603 LDA      45       Variablen-Anfangspointer
42605 LDY      46
42607 STA      47       Array-Anfangspointer
42609 STY      48
42611 STA      49       Variablen-Endepointer
42613 STY      50
42615 JSR   43037       > RESTORE
42618 LDX     #25       Stringdescriptorzeiger ruecksetzen
42620 STX      22
42622 PLA               Ruecksprungadresse merken
42623 TAY
42624 PLA
42625 LDX    #250       Stack ruecksetzen
42627 TXS
42628 PHA               Ruecksprungadresse wiederherstellen
42629 TYA
42630 PHA
42631 LDA      #0
42633 STA      62       CONT sperren
42635 STA      16       Flag fuer Variablenverwaltung
42637 RTS
```

CHRGET-Pointer ruecksetzen

```
42638 CLC
42639 LDA      43       Programm-Anfangspointer low
42641 ADC    #255       - 1
42643 STA     122       ergibt CHRGET-Pointer low
42645 LDA      44       Programm-Anfangspointer high
42647 ADC    #255       + Uebertrag
42649 STA     123       ergibt CHRGET-Pointer high
42651 RTS
```

BASIC-Routine LIST

```
42652 BCC  42660      erstes Zeichen Ziffer oder
42654 BEQ  42660      Trennungszeichen? Ja: weiter bei 42660
42656 CMP   #171      folgt "-" nach 'LIST'?
42658 BNE  42637      Nein: SYNTAX ERROR
42660 JSR  43371      > erste Zeilennummer nach (20/21) (= 0, falls "-" folgt)
42663 JSR  42515      > Startadresse der Zeile nach (95/96)
42666 JSR    121      > CHRGOT holt letztes Zeichen
42669 BEQ  42683      Trennungszeichen? Ja: weiter bei 42683
42671 CMP   #171      "-"?
42673 BNE  42561      Nein: SYNTAX ERROR
42675 JSR    115      > CHRGET holt naechstes Zeichen
42678 JSR  43371      > zweite Zeilennummer nach (20/21)
42681 BNE  42561      Folgt Trennungszeichen? Nein: "SYNTAX ERROR"
42683 PLA             Ruecksprungadresse in Interpreterschleife entfernen
42684 PLA
42685 LDA     20      zweite Zeilennummer = 0? (oder Zeilennummer, falls nur
42687 ORA     21      eine Nummer angegeben; daher funktioniert 'LIST 0' nicht)
42689 BNE  42697      Nein: weiter bei 42697
42691 LDA   #255
42693 STA     20      zweite Zeilennummer := 65535 (Maximalwert)
42695 STA     21

42697 LDY     #1      Indirect-Pointer setzen
42699 STY     15      Quote-Modus abschalten
42701 LDA    (95),Y   Linkadresse high = 0 (Programmende)?
42703 BEQ  42772      Ja: fertig, weiter bei 42772 (READY.)
42705 JSR  43052      > RUNSTOP-Taste abfragen, evtl. Abbruch
42708 JSR  43735      > Zeilenvorschub drucken
42711 INY
42712 LDA    (95),Y   Zeilenummer nach (XR/Accu)
42714 TAX
42715 INY
42716 LDA    (95),Y
42718 CMP     21      Ende erreicht?
42720 BNE  42726
42722 CPX     20
42724 BEQ  42728
42726 BCS  42772      Ja: fertig, weiter bei 42772
42728 STY     73      YR zwischenspeichern
42730 JSR  48589      > Zeilennummer in String umwandeln und drucken
```

BASIC-Routine LIST (Fortsetzung)

```
42733 LDA    #32       Code fuer Space
42735 LDY     73       YR wiederherstellen
42737 AND   #127       Bit 7 loeschen
42739 JSR  43847       > Zeichen drucken
42742 CMP    #34       Anfuehrungszeichen?
42744 BNE  42752       Nein: weiter bei 42752
42746 LDA     15       Quote-Flag invertieren
42748 EOR   #255
42750 STA     15
42752 INY
42753 BEQ  42772       Kein Ende der Zeile nach 255 Zeichen? Ja: Aufhoeren!
42755 LDA    (95),Y    Naechstes Zeichen
42757 BNE  42775       Zeilenende? Nein: weiter bei 42775
42759 TAY              YR := 0
42760 LDA    (95),Y    Startadresse der naechsten Zeile
42762 TAX
42763 INY
42764 LDA    (95),Y
42766 STX     95       nach (95/96) bringen
42768 STA     96
42770 BNE  42697       Unbedingter Sprung

42772 JMP  58246       > Ausgabe "READY.", Eingabewarteschleife

42775 JMP   (774)      Normalwert des Vektors (774/775): 42778
42778 BPL  42739       Code < 128? Ja: zurueck nach 42739
42780 CMP   #255       Code fuer PI?
42782 BEQ  42739       Ja: zurueck nach 42739
42784 BIT     15       Quote-Modus eingeschaltet?
42786 BMI  42739       Ja: zurueck nach 42739
42788 SEC
42789 SBC   #127
42791 TAX              Wortzaehler fuer Befehlsworttabelle
42792 STY     73       YR zwischenspeichern
42794 LDY   #255       Pointer auf Befehlstabelle
42796 DEX              vermindern. Klartext fuer Token gefunden?
42797 BEQ  42807       Ja: Ausdrucken, weiter 42807
42799 INY
42800 LDA  41118,Y     Zeichen in Befehlsworttabelle
42803 BPL  42799       ueberlesen, bis
42805 BMI  42796       letztes Zeichen des Befehlswortes erreicht
42807 INY
42808 LDA  41118,Y     Zeichen aus Befehlsworttabelle
42811 BMI  42735       letztes Zeichen im Wort? Ja: zurueck nach 42735
42813 JSR  43847       > Zeichen drucken
42816 BNE  42807       Im Normalfall unbedingter Sprung (ausser Code 204)
```

BASIC-Routine FOR

```
42818 LDA   #128     Bit 7 in (16) setzen, um Annahme von Integer- und Feld-
42820 STA     16     variablen als Schleifenvariable zu verhindern
42822 JSR  43429     > LET definiert FOR-NEXT-Variable, Pointer nach (73/74)
42825 JSR  41866     > Stack-Suchroutine sucht nach offener FOR-NEXT-Schleife
42828 BNE  42835       mit dieser Variablen. Gefunden? Nein: weiter bei 42835
42830 TXA            Stackpointer auf alte Schleife setzen
42831 ADC    #15
42833 TAX
42834 TXS
42835 PLA            Ruecksprungadresse Interpreterschleife entfernen
42836 PLA
42837 LDA     #9     Zur Pruefung, ob Platz im Stack
42839 JSR  41979     > Wenn Stackpointer < 80, dann OUT OF MEMORY
42842 JSR  43270     > naechstes Trennzeichen suchen, Offset im YR
42845 CLC
42846 TYA            Offset zum CHRGET-Pointer addieren,
42847 ADC    122     ergibt Startadresse der FOR-NEXT-Schleife
42849 PHA            Startadresse low auf den Stack ablegen
42850 LDA    123
42852 ADC     #0     Uebertrag addieren
42854 PHA            ebenso high auf den Stack legen
42855 LDA     58     Aktuelle Zeilennummer low
42857 PHA            und
42858 LDA     57     Aktuelle Zeilennummer high
42860 PHA            auf Stack legen
42861 LDA   #164     BASIC-Code fuer 'TO'
42863 JSR  44799     > SYNCHR prueft, ob dieser Code folgt
42866 JSR  44429     > FRMNUM prueft, ob numerische Schleifenvariable folgte
42869 JSR  44426     > FRMEVL wertet Ausdruck aus, Ergebnis nach FAC
42872 LDA    102     Vorzeichenbyte von FAC (0/255 fuer +/-)
42874 ORA   #127     Bit 0 bis 6 fuer AND setzen
42876 AND     98     Bit 0 bis 6 im Accu mit Bits 0 bis 6 von FAC gleichmachen
42878 STA     98     Ergebnis als MSB speichern, Bit 7 ergibt Vorzeichen
42880 LDA   #139     Ruecksprungadresse 42891 fuer indirekten Sprung
42882 LDY   #167     (vgl. 44629)
42884 STA     34     nach (34/35) bringen
42886 STY     35
42888 JMP  44611     > TO-Wert auf Stack ablegen, dann zurueck nach 42891
42891 LDA   #188     (Accu/YR) := 47548, Startadresse von
42893 LDY   #185     Ersatzwert 1 fuer 'STEP'
42895 JSR  48034     > 1 als STEP-Wert nach FAC bringen
42898 JSR    121     > CHRGOT holt letztes Zeichen
42901 CMP   #169     BASIC-Code fuer 'STEP'?
42903 BNE  42911     Nein: kein STEP-Wert angegeben, weiter bei 42911
42905 JSR    115     > CHRGET holt naechstes Zeichen
42908 JSR  44426     > FRMEVL wertet Ausdruck aus, Ergebnis nach FAC
42911 JSR  48171     > holt Vorzeichenbyte
42914 JSR  44600     > legt Vorzeichen und STEP-Wert auf Stack
42917 LDA     74     FOR-NEXT-Variablenzeiger low,
42919 PHA
42920 LDA     73     FOR-NEXT-Variablenzeiger high
42922 PHA            und
42923 LDA   #129     abschliessend noch den Code fuer 'FOR'
42925 PHA            auf den Stack legen
```

Interpreter-Schleife, Routinenaufruf

```
42926 JSR   43052        > RUNSTOP-Taste abfragen, evtl. Abbruch
42929 LDA     122        CHRGET-Pointer low,
42931 LDY     123        CHRGET-Pointer high
42933 CPY      #2        = 2 (Direktmodus)?
42935 NOP
42936 BEQ   42942        Ja: weiter bei 42942
42938 STA      61        Zeiger fuer CONT
42940 STY      62
42942 LDY      #0
42944 LDA   (122),Y      laufendes Zeichen
42946 BNE   43015        = 0? Nein: weiter bei 43015
42948 LDY      #2
42950 LDA   (122),Y      uebernaechstes Zeichen
42952 CLC
42953 BNE   42958        = 0 (Programmende)? Nein: weiter bei 42958
42955 JMP   43083        > Programmlauf beenden

42958 INY
42959 LDA   (122),Y      naechste Zeilennummer low
42961 STA      57        nach (57)
42963 INY
42964 LDA   (122),Y      naechste Zeilennummer high
42966 STA      58        nach (58)
42968 TYA
42969 ADC     122        CHRGET-Pointer auf letztes Byte nach
42971 STA     122        neuem Zeilenkopf setzen
42973 BCC   42977
42975 INC     123
42977 JMP   (776)        Normalwert des Vektors (776/777): 42980
42980 JSR     115        > CHRGET holt naechstes Zeichen
42983 JSR   42989        > Interpretation und Routinenaufruf
42986 JMP   42926        > Zurueck zum Schleifenanfang

42989 BEQ   43051        Trennzeichen? Ja: RTS
42991 SBC    #128        128 vom Zeichencode abziehen
42993 BCC   43012        < 0 (kein BASIC-Befehl)? Ja: LET, weiter bei 43012
42995 CMP     #35        Code > 162 (NEW)? (162 - 128 = 35)
42997 BCS   43022        Ja: weiter bei 43022
42999 ASL                Ergebnis verdoppeln
43000 TAY                als Pointerins YR bringen
43001 LDA   40973,Y      Adresse high aus Tabelle
43004 PHA                auf Stack legen
43005 LDA   40972,Y      Adresse low aus Tabelle
43008 PHA                auf Stack legen
43009 JMP     115        > CHRGET holt naechstes Zeichen, RTS ruft Routine auf

43012 JMP   43429        > LET, weist Variablen einen Wert zu

43015 CMP     #58        Trennzeichen ":"?
43017 BEQ   42977        Ja: zurueck nach 42997
43019 JMP   44808        > "SYNTAX ERROR"
43022 CMP     #75        BASIC-Code fuer 'GO'?
43024 BNE   43019        Nein: "SYNTAX ERROR"
43026 JSR     115        > CHRGET holt naechstes Zeichen
43029 LDA    #164        BASIC-Code fuer 'TO'
43031 JSR   44799        > SYNCHR prueft, ob dieser Code folgt
43034 JMP   43168        > GOTO
```

BASIC-Routine RESTORE

```
43037 SEC
43038 LDA      43       vom Programm-Anfangspointer
43040 SBC      #1       eins subtrahieren
43042 LDY      44
43044 BCS   43047
43046 DEY
43047 STA      65       ergibt DATA-Zeiger
43049 STY      66
43051 RTS
```

BASIC-Routinen END und STOP sowie Abfrage der RUNSTOP-Taste mit Abbruch

```
43052 JSR   65505       > STOP prueft RUNSTOP-Taste
43055 BCS   43058       Einsprung fuer 'STOP'
43057 CLC               Einsprung fuer 'END'
43058 BNE   43120       RUNSTOP-Taste nicht gedrueckt? Ja: RTS
43060 LDA     122       CHRGET-Pointer low,
43062 LDY     123       CHRGET-Pointer high
43064 LDX      58       Zeilennummer high <255 im Direktmodus>?
43066 INX
43067 BEQ   43081       Direktmodus? Ja: weiter bei 43081
43069 STA      61       CHRGET-Pointer nach <61/62>
43071 STY      62
43073 LDA      57       aktuelle Zeilennummer fuer CONT
43075 LDY      58
43077 STA      59       in <59/60> aufbewahren
43079 STY      60
43081 PLA               Ruecksprungadresse aus Stack entfernen
43082 PLA
43083 LDA    #129       <Accu/YR> := 51857, Startadresse fuer "BREAK"
43085 LDY    #163
43087 BCC   43092       Aufruf durch END? Ja: keine Ausgabe von "BREAK"
43089 JMP   42089       > "BREAK"-Meldung, READY.

43092 JMP   58246       > Ausgabe "READY.", Eingabewarteschleife
```

BASIC-Routine CONT

```
43095 BNE   43120       Folgt Trennungszeichen? Nein: "SYNTAX ERROR"
43097 LDX     #26       Code fuer "CAN'T CONTINUE"
43099 LDY      62       'CONT' gesperrt?
43101 BNE   43106       Nein: weiter bei 43106
43103 JMP   42039       > Fehlermeldung, READY.

43106 LDA      61
43108 STA     122       CHRGET-Pointer wiederherstellen
43110 STY     123
43112 LDA      59
43114 LDY      60
43116 STA      57       Zeilennummer wiederherstellen
43118 STY      58
43120 RTS
```

BASIC-Routine RUN

```
43121 PHP             Statusregister merken
43122 LDA     #0
43124 JSR  65424      > SETMSG, Ausgabemodus festlegen
43127 PLP
43128 BNE  43133      folgt Trennungszeichen? Nein: RUN mit Zeilennummer
43130 JMP  42585      > CHRGET-Pointer ruecksetzen, CLR

43133 JSR  42592      > CLR
43136 JMP  43159      > CHRGOT, GOTO, Interpreter-Schleife
```

BASIC-Routine GOSUB

```
43139 LDA     #3      Zur Pruefung, ob Platz im Stack
43141 JSR  41979      > Wenn Stackpointer < 68, dann "OUT OF MEMORY"
43144 LDA    123      CHRGET-Pointer high,
43146 PHA
43147 LDA    122      CHRGET-Pointer low,
43149 PHA
43150 LDA     58      Zeilennummer high,
43152 PHA
43153 LDA     57      Zeilennummer low,
43155 PHA             und
43156 LDA   #141      abschliessend noch den Code fuer 'GOSUB'
43158 PHA             auf den Stack
43159 JSR    121      > CHRGET holt naechstes Zeichen
43162 JSR  43168      > GOTO
43165 JMP  42926      > Interpreter-Schleife
```

BASIC-Routine GOTO

```
43168 JSR  43371      > Zeilennummer hinter GOTO nach (20/21) bringen
43171 JSR  43273      > Offset bis zum Ende der Zeile ins YR
43174 SEC
43175 LDA     57      Aktuelle Zeilennummer mit Zeilennummer
43177 SBC     20      nach 'GOTO' vergleichen
43179 LDA     58
43181 SBC     21
43183 BCS  43196      Groesser: weiter bei 43196
43185 TYA
43186 SEC             Offset zum Zeilenende zum CHRGET-Pointer addieren
43187 ADC    122
43189 LDX    123
43191 BCC  43200
43193 INX
43194 BCS  43200      Unbedingter Sprung

43196 LDA     43      (Accu/XR) := Pointer auf Programmanfang
43198 LDX     44
43200 JSR  42519      > Startadresse der gesuchten Zeile nach (95/96)
43203 BCC  43235      Zeile vorhanden? Nein: "UNDEF'D STATEMENT ERROR"
43205 LDA     95      von der Startadresse der Zeile
43207 SBC     #1      eins subtrahieren
43209 STA    122
43211 LDA     96
43213 SBC     #0
43215 STA    123      ergibt neuen CHRGET-Pointer, zeigt auf Null-Code
43217 RTS
```

BASIC-Routine RETURN

```
43218 BNE  43217       folgt Trennzeichen? Nein: "SYNTAX ERROR"
43220 LDA   #255
43222 STA     74       FOR-NEXT-Variablenzeiger high zwangsweise definieren
43224 JSR  41866       > Stack-Suchroutine sucht nach naechstem GOSUB-Datensatz
43227 TXS
43228 CMP   #141       'GOSUB'-Code gefunden?
43230 BEQ  43243       Ja: weiter bei 43243
43232 LDX    #12       Code fuer "RETURN WITHOUT GOSUB"
43234 BIT    ...
43235 LDX    #17       Code fuer "UNDEF'D STATEMENT"
43237 JMP  42039       > Fehlermeldung, READY.

43240 JMP  44808       > "SYNTAX ERROR"

43243 PLA              'GOSUB'-Code vom Stack holen
43244 PLA              Zeilennummer vom Stack
43245 STA     57       nach (57/58) bringen
43247 PLA
43248 STA     58
43250 PLA              CHRGET-Pointer vom Stack
43251 STA    122       nach (122/123) bringen
43253 PLA
43254 STA    123
```

BASIC-Routine DATA

```
43256 JSR  43270       > Offset zum naechsten Trennzeichen feststellen
43259 TYA
43260 CLC              CHRGET-Pointer auf naechstes Trennzeichen setzen
43261 ADC    122
43263 STA    122
43265 BCC  43269
43267 INC    123
43269 RTS
```

Offset zum naechsten Trennzeichen feststellen

```
43270 LDX    #58       Trennzeichen zwischen Befehlen
43272 BIT    ...
43273 LDX     #0       Trennzeichen zwischen Zeilen
43275 STX      7       abspeichern
43277 LDY     #0       Zaehler initialisieren
43279 STY      8       (8) := 0
43281 LDA      8
43283 LDX      7
43285 STA      7
43287 STX      8
43289 LDA   (122),Y    Zeichen holen
43291 BEQ  43269       Zeilenende? Ja: RTS
43293 CMP      8       = (8)?
43295 BEQ  43269       Ja: gegebenes Trennzeichen gefunden, RTS
43297 INY              Index erhoehen
43298 CMP    #34       Anfuehrungszeichen?
43300 BNE  43289       Nein: weitersuchen ...
43302 BEQ  43281       Sonst erst (7) und (8) vertauschen
```

BASIC-Routine IF

```
43304 JSR  44446      > FRMEVL wertet Ausdruck aus, Ergebnis nach FAC
43307 JSR    121      > CHRGOT holt letztes Zeichen
43310 CMP   #137      = BASIC-Code fuer 'GOTO'?
43312 BEQ  43319      Ja: weiter bei 43319
43314 LDA   #167      BASIC-Code fuer 'THEN'
43316 JSR  44799      > SYNCHR prueft, ob dieser Code folgt
43319 LDA     97      IF-Bedingung erfuellt?
43321 BNE  43328      Ja: weiter bei 43328
43323 JSR  43273      > Offset des Zeilenendes ins YR bringen
43326 BEQ  43259      CHRGET-Pointer auf naechste Zeile setzen
43328 JSR    121      > CHRGOT holt letztes Zeichen
43331 BCS  43336      Ziffer? Nein: kein Sprungbefehl
43333 JMP  43168      > GOTO

43336 JMP  42989      > Interpreter-Schleife
```

BASIC-Routine ON

```
43339 JSR  47006      > GETBYT bringt Wert von 0 bis 255 nach (101)
43342 PHA             letztes Zeichen merken
43343 CMP   #141      Code fuer 'GOSUB'?
43345 BEQ  43351      Ja: weiter bei 43351
43347 CMP   #137      Code fuer 'GOTO'?
43349 BNE  43240      Nein: "SYNTAX ERROR"
43351 DEC    101      Variablenwert vermindern
43353 BNE  43359      Ergebnis = 0? Nein: weiter bei 43359
43355 PLA             letztes Zeichen vom Stack holen
43356 JMP  42991      > Interpreter-Schleife, Routinenaufruf

43359 JSR    115      > CHRGET holt naechstes Zeichen
43362 JSR  43371      > Zeilennummer nach (20/21) bringen
43365 CMP    #44      Folgt Komma?
43367 BEQ  43351      Ja: weiter bei 43351
43369 PLA             keine Sprungadresse vorhanden
43370 RTS
```

Lesen einer Zeilennummer und Umwandlung in 16-Bit nach (20/21)

Eingabe: laufendes CHRGET-Zeichen im Accu, zugehoeriger Wert im Statusregister

Ausgabe: a) wenn erstes Zeichen numerisch:
            Wert der Zeilennummer bis zum ersten nichtnumerischen
            Zeichen in 16-Bit-Integer in (20/21)
         b) wenn erstes Zeichen nicht numerisch:
            (20/21) := 0

In beiden Faellen ist die Carry-Flag gesetzt

```
43371 LDX     #0
43373 STX     20       (20/21) mit 0 vorbesetzen
43375 STX     21
43377 BCS  43370       Zeichen numerisch? Nein: RTS
43379 SBC    #47       vom ASCII den Wert 48 subtrahieren (Carry = 1)
43381 STA      7       Ziffernwert merken
43383 LDA     21
43385 STA     34
43387 CMP    #25       Ergebnis im zulaessigen Bereich (0-63999)?
43389 BCS  43347       Nein: im Normalfall "SYNTAX ERROR"
43391 LDA     20       bisher gelesene Zahl
43393 ASL              mit 10 multiplizieren
43394 ROL     34
43396 ASL              zweimal mal 2,
43397 ROL     34
43399 ADC     20
43401 STA     20
43403 LDA     34       plus vorherigen Wert,
43405 ADC     21
43407 STA     21
43409 ASL     20       und dann verdoppeln
43411 ROL     21
43413 LDA     20
43415 ADC      7       Ziffernwert addieren
43417 STA     20
43419 BCC  43423
43421 INC     21       Uebertrag addieren
43423 JSR    115       > CHRGET holt naechstes Zeichen
43426 JMP  43377       > Zurueck zum Schleifenanfang
```

BASIC-Routine LET

```
43429 JSR  45195       > VARSUC sucht Variable hinter LET, oder legt sie an
43432 STA    73        Variablenzeiger nach (73/74) bringen
43434 STY    74
43436 LDA   #178       Code fuer "="
43438 JSR  44799       > SYNCHR prueft, ob dieser Code folgt
43441 LDA    14        Integer-Flag und
43443 PHA
43444 LDA    13        String-Flag auf Stack retten
43446 PHA
43447 JSR  44446       > FRMEVL wertet Ausdruck aus
43450 PLA
43451 ROL              Bei String ist die Carry-Flag gesetzt, sonst geloescht
43452 JSR  44432       > FRMNUM prueft, ob Ergebnis zum Variablentyp passt
43455 BNE  43481       String? Ja: weiter bei 43481
43457 PLA              Integer?
43458 BPL  43478       Nein: weiter bei 43478
43460 JSR  48155       > FAC runden
43463 JSR  45503       > FLPINT wandelt Fliesskommazahl in 16-Bit-Integer
43466 LDY    #0
43468 LDA   100        Byte high in Variable uebertragen,
43470 STA   (73),Y
43472 INY
43473 LDA   101        Byte low in Variable uebertragen
43475 STA   (73),Y
43477 RTS

43478 JMP  48080       > Fliesskommazahl aus FAC in Variable uebertragen

43481 PLA
43482 LDY    74        Variablenzeiger high
43484 CPY   #191       = 191 (TI$, vgl. 45356 ff)?
43486 BNE  43564       Nein: weiter bei 43564
43488 JSR  46758       > FRESTR
43491 CMP    #6        Stringlaenge = 6?
43493 BNE  43556       Nein: "ILLEGAL QUANTITY ERROR"
43495 LDY    #0
43497 STY    97        FAC initialisieren
43499 STY   102
43501 STY   113
43503 JSR  43549       > prueft auf Ziffer, addiert Wert zu FAC
43506 JSR  47842       > FAC := 10 * FAC
43509 INC   113        Stellenzaehler erhoehen
43511 LDY   113
43513 JSR  43549       > prueft auf Ziffer, addiert Wert zu FAC
43516 JSR  48140       > ARG := FAC
43519 TAX
43520 BEQ  43527       FAC = 0? Ja: weiter bei 43527
43522 INX
43523 TXA              Exponent von FAC erhoehen (entspricht FAC := FAC * 2)
43524 JSR  47853       > FAC := (ARG + FAC) * 2
43527 LDY   113        also insgesamt FAC := 6 * FAC
43529 INY              Stellenzaehler erhoehen
43530 CPY    #6        = 6?
43532 BNE  43501       Nein: weitermachen ...
43534 JSR  47842       > FAC := 10 * FAC
43537 JSR  48283       > Fliesskomma in Integer umwandeln
43540 LDX   100
43542 LDY    99
43544 LDA   101
43546 JMP  65499       > SETTIM, Uhrzeit setzen
```

BASIC-Routine LET (Fortsetzung)

```
43549 LDA   (34),Y    Zeichen aus String holen
43551 JSR    128      > pruefen, ob Ziffer
43554 BCC  43559      Ja: weiter bei 43559
43556 JMP  45640      > "ILLEGAL QUANTITY ERROR"
43559 SBC    #47      48 (Carry = 0) subtrahieren
43561 JMP  48510      > Accu zu FAC addieren

43564 LDY     #2
43566 LDA  (100),Y    Startadresse high des String unterhalb des
43568 CMP     52      Stringbereichs (also im Programm)?
43570 BCC  43595      Ja: weiter bei 43595
43572 BNE  43581      Groesser? Ja: weiter bei 43581
43574 DEY
43575 LDA  (100),Y    Startadresse low des String
43577 CMP     51      mit String-Anfangszeiger low vergleichen
43579 BCC  43595      kleiner (also im Programm)? Ja: weiter bei 43595
43581 LDY    101      Pointer high auf Stringdescriptor
43583 CPY     46      mit Pointer high auf Anfang der Variablen vergleichen
43585 BCC  43595      kleiner? Ja: weiter bei 43595
43587 BNE  43602      Groesser? Ja: weiter bei 43602
43589 LDA    100      Pointer low auf Stringdescriptor
43591 CMP     45      mit Pointer low auf Anfang der Variablen vergleichen
43593 BCS  43602      Groesser? Ja: weiter bei 43602
43595 LDA    100      (Accu/YR) := Zeiger auf Stringdescriptor
43597 LDY    101
43599 JMP  43624      > bis 43624 ueberspringen
43602 LDY     #0
43604 LDA  (100),Y    Stringlaenge
43606 JSR  46197      > Speicherplatz pruefen, Stringpointer setzen
43609 LDA     80      (80/81) zeigt auf Stringdescriptor
43611 LDY     81
43613 STA    111
43615 STY    112
43617 JSR  46714      > String in den Stringbereich uebertragen
43620 LDA    #97      Stringdescriptor steht in (97,...,99)
43622 LDY     #0
43624 STA     80
43626 STY     81
43628 JSR  46811      > mit Pointer auf letzten Descriptor vergleichen
43631 LDY     #0
43633 LDA   (80),Y    Stringdescriptor als Stringvariablenwert in
43635 STA   (73),Y    Variablentabelle bringen
43637 INY
43638 LDA   (80),Y
43640 STA   (73),Y
43642 INY
43643 LDA   (80),Y
43645 STA   (73),Y
43647 RTS
```

BASIC-Routine PRINT#

```
43648 JSR  43654       > CMD
43651 JMP  43957       > CLRCH, Tastatur aktivieren
```

BASIC-Routine CMD

```
43654 JSR  47006       > GETBYT holt Filenummer
43657 BEQ  43664       Folgt Trennzeichen? Ja: weiter bei 43664
43659 LDA   #44        Komma-Code
43661 JSR  44799       > SYNCHR prueft, ob dieser Code folgt
43664 PHP              Statusregister fuer PRINT merken
43665 STX    19        aktiver Ausgabe-Kanal
43667 JSR  57624       > CHKOUT mit Fehlerbehandlung
43670 PLP              Statusregister wiederherstellen
43671 JMP  43680       > PRINT
```

BASIC-Routine PRINT

```
43674 JSR  43809       > String drucken
43677 JSR   121        > CHRGOT holt letztes Zeichen
```

Einsprung fuer PRINT

```
43680 BEQ  43735       Trennzeichen? Ja: weiter bei 43735
43682 BEQ  43751       Trennzeichen (nach 'TAB' und 'SPC')? Ja: RTS
43684 CMP  #163        BASIC-Code fuer 'TAB'?
43686 BEQ  43768       Ja: weiter bei 43768
43688 CMP  #166        BASIC-Code fuer 'SPC'?
43690 CLC              Flag fuer 'SPC' setzen
43691 BEQ  43768       'SPC'-Code? Ja: weiter bei 43768
43693 CMP   #44        Komma-Code?
43695 BEQ  43752       Ja: weiter bei 43752
43697 CMP   #59        Code fuer Semikolon?
43699 BEQ  43795       Ja: naechstes Zeichen lesen, Schleifenanfang
43701 JSR  44446       > FRMEVL wertet Ausdruck aus
43704 BIT    13        String?
43706 BMI  43674       Ja: String drucken, Schleifenanfang
43708 JSR  48605       > FACSTR, wandelt FAC in String ab (256) um
43711 JSR  46215       > String-Parameter holen
43714 JSR  43809       > String drucken
43717 JSR  43835       > 'CURSOR RIGHT' oder 'SPACE' drucken
43720 BNE  43677       Unbedingter Sprung zum Schleifenanfang
```

Abschluss der Eingabewarteschleife

```
43722 LDA    #0        Eingabe-Puffer mit 0 abschliessen
43724 STA   512,X
43727 LDX  #255        CHRGET-Pointer auf 511 setzen
43729 LDY    #1
43731 LDA    19        Aktiver Eingabe-Kanal = Tastastur?
43733 BNE  43751       Nein: RTS
```

BASIC-Routine PRINT (Fortsetzung)

```
43735 LDA   #13        Code fuer 'RETURN'
43737 JSR  43847       > Zeilenvorschub drucken
43740 BIT    19        Aktiver Eingabe-Kanal < 128?
43742 BPL  43749       Ja: fertig
43744 LDA   #10        Code fuer 'LINEFEED'
43746 JSR  43847       > Zeichen drucken
43749 EOR  #255
43751 RTS
```

Sprung zur naechsten vortabulierten Position

```
43752 SEC
43753 JSR  65520      > PLOT, Cursorposition lesen, Spalte im YR
43756 TYA
43757 SEC
43758 SBC   #10       davon solange 10 subtrahieren,
43760 BCS  43758      bis Ergebnis negativ
43762 EOR  #255       Vorzeichen umdrehen
43764 ADC    #1       (Zweierkomplement plus eins)
43766 BNE  43790      Unbedingter Sprung
```

Ausfuehrung von TAB und SPC

```
43768 PHP             Flag fuer SPC (Carry = 0) merken
43769 SEC
43770 JSR  65520      > PLOT, Cursorposition lesen, Spalte im YR
43773 STY     9       Spalte merken
43775 JSR  47003      > GETBYT holt TAB/SPC-Parameter ins XR
43778 CMP   #41       Folgt ")"?
43780 BNE  43871      Nein: "SYNTAX ERROR"
43782 PLP             Carry holen
43783 BCC  43791      SPC? Ja: weiter bei 43791
43785 TXA             TAB-Parameter
43786 SBC     9       kleiner als aktuelle Cursorspalte?
43788 BCC  43795      Ja: nichts ausgeben, fertig
43790 TAX             Anzahl der Schritte bis zum Tabulator ins XR
43791 INX             XR als Zaehler initialisieren
43792 DEX             XR vermindern
43793 BNE  43801      = 0? Nein: 'CURSOR RIGHT' drucken, zurueck nach 43792
43795 JSR   115       > CHRGET holt naechstes Zeichen
43798 JMP  43682      > Sprung zum Anfang der PRINT-Schleife
43801 JSR  43835      > 'CURSOR RIGHT' oder 'SPACE' drucken
43804 BNE  43792      Unbedingter Sprung

43806 JSR  46215      > Stringparameter holen
43809 JSR  46758      > FRESTR
43812 TAX             Stringlaenge
43813 LDY    #0       Indirect-Pointer fuer Stringausgabe
43815 INX
43816 DEX
43817 BEQ  43751      fertig gedruckt? Ja: RTS
43819 LDA   (34),Y    Zeichen aus String holen
43821 JSR  43847      > drucken
43824 INY
43825 CMP   #13       War Zeichen ein 'CR'?
43827 BNE  43816      Nein: zurueck nach 43816
43829 JSR  43749      > EOR #255 (?)
43832 JMP  43816      > zurueck zum Schleifenanfang
43835 LDA    19       Ausgabe ueber logisches File?
43837 BEQ  43842      Nein: 'CURSOR RIGHT' statt 'SPACE' drucken
43839 LDA   #32       Code fuer 'SPACE'
43841 BIT   ...
43842 LDA   #29       Code fuer 'CURSOR RIGHT'
43844 BIT   ...
43845 LDA   #63       Code fuer "?"
43847 JSR  57612      > CHROUT mit Fehlerbehandlung
43850 AND  #255       Statusflags setzen
43852 RTS
```

Fehlerbehandlung bei INPUT, GET und READ

```
43853 LDA     17       Flag fuer INPUT (=0), GET (=64), READ (=152)
43855 BEQ  43874       INPUT: weiter bei 43874
43857 BMI  43863       READ: weiter bei 43863
43859 LDY   #255       GET:
43861 BNE  43867       Unbedingter Sprung
43863 LDA     63       DATA-Zeilennummer
43865 LDY     64
43867 STA     57       als Zeilennummer fuer Fehlermeldung speichern
43869 STY     58
43871 JMP  44808       > "SYNTAX ERROR"

43874 LDA     19       Eingabe ueber File?
43876 BEQ  43883       Nein: "REDO FROM START"
43878 LDX    #24       Code fuer "FILE DATA"
43880 JMP  42039       > Fehlermeldung, READY.

43883 LDA    #12       (Accu/YR) := 44300, Startadresse fuer "REDO FROM START"
43885 LDY   #173
43887 JSR  43806       > Fehlermeldung drucken
43890 LDA     61       CHRGET-Pointer auf INPUT zuruecksetzen
43892 LDY     62
43894 STA    122
43896 STY    123
43898 RTS
```

BASIC-Routine GET

```
43899 JSR  45990       > Falls Direktmodus, "ILLEGAL DIRECT ERROR"
43902 CMP    #35       Folgt "#"?
43904 BNE  43922       Nein: weiter bei 43922
43906 JSR    115       > CHRGET holt naechstes Zeichen
43909 JSR  47006       > GETBYT holt Filenummer ins XR
43912 LDA    #44       Komma-Code
43914 JSR  44799       > SYNCHR prueft, ob dieser Code folgt
43917 STX     19       Filenummer
43919 JSR  57630       > CHKIN, Eingabevorbereitung
43922 LDX     #1       Eingabezeiger auf 513 setzen
43924 LDY     #2
43926 LDA     #0
43928 STA    513       Endezeichen nach 513, da nur ein Zeichen geholt wird
43931 LDA    #64       Flag-Wert fuer GET
43933 JSR  44047       > holt Zeichen und bringt Code in Variable
43936 LDX     19       Eingabe ueber File?
43938 BNE  43959       Ja: CLRCHN, Tastatur aktivieren
43940 RTS
```

BASIC-Routine INPUT#

```
43941 JSR  47006       > GETBYT bringt Filenummer ins XR
43944 LDA    #44       Komma-Code
43946 JSR  44799       > SYNCHR prueft, ob dieser Code folgt
43949 STX     19       Filenummer
43951 JSR  57630       > CHKIN, Eingabevorbereitung
43954 JSR  43982       > INPUT ohne Dialog
43957 LDA     19       Filenummer in Accu
43959 JSR  65484       > CLRCHN, aktive I/O-Kanaele schliessen
43962 LDX     #0
43964 STX     19       Tastatur aktivieren
43966 RTS
```

BASIC-Routine INPUT

```
43967 CMP    #34       Folgt Anfuehrungszeichen?
43969 BNE  43982       Nein: weiter bei 43982
43971 JSR  44733       > Dialogstring erfassen
43974 LDA    #59       Code fuer Semikolon
43976 JSR  44799       > SYNCHR prueft, ob dieser Code folgt
43979 JSR  43809       > Dialogstring drucken
43982 JSR  45990       > Falls Direktmodus, "ILLEGAL DIRECT ERROR"
43985 LDA    #44       Komma-Code
43987 STA    511       nach 511
43990 JSR  44025       > evtl. "?" drucken, Uebergabe an Eingabewarteschleife
43993 LDA     19       Eingabe ueber File?
43995 BEQ  44010       Nein: weiter bei 44010
43997 JSR  65463       > READST, Status lesen
44000 AND     #2       Bit 1 (Timeout Read) isolieren
44002 BEQ  44010       geloescht? Ja: weiter bei 44010
44004 JSR  43957       > CLRCHN, Tastatur aktivieren
44007 JMP  43256       > DATA, Programm nach naechstem Trennzeichen forsetzen

44010 LDA    512       Erstes Zeichen aus Eingabepuffer = 0 (keine Eingabe)?
44013 BNE  44045       Nein: Eingabetext durch 'READ' verarbeiten
44015 LDA     19       Eingabe ueber File?
44017 BNE  43990       Ja: nochmal probieren, zurueck nach 43990
44019 JSR  43270       > Offset zum naechsten Trennzeichen feststellen
44022 JMP  43259       > CHRGET-Pointer auf naechsten Befehl setzen

44025 LDA     19       Eingabe ueber File?
44027 BNE  44035       Ja: 'INPUT' ohne "?", weiter bei 44035
44029 JSR  43845       > Fragezeichen drucken
44032 JSR  43835       > 'SPACE' drucken
44035 JMP  42336       > Eingabewarteschleife
```

BASIC-Routine READ

```
44038 LDX     65       DATA-Zeiger
44040 LDY     66
44042 LDA   #152       Flagwert fuer READ
44044 BIT    ...
44045 LDA     #0       Flagwert fuer INPUT
44047 STA     17       Flag setzen
44049 STX     67       Eingabezeiger auf Eingabequelle setzen
44051 STY     68
44053 JSR  45195       > VARSUC sucht Variable oder legt sie an
44056 STA     73       Variablenzeiger nach (73/74)
44058 STY     74
44060 LDA    122       CHRGET-Pointer
44062 LDY    123
44064 STA     75       nach (75/76) bringen
44066 STY     76
44068 LDX     67       Eingabezeiger
44070 LDY     68
44072 STX    122       nach CHRGET-Pointer bringen
44074 STY    123
44076 JSR    121       > CHRGOT holt Zeichen aus Eingabequelle
44079 BNE  44113       Endezeichen? Nein: weiter bei 44113
44081 BIT     17       INPUT/GET/READ-Flag
44083 BVC  44097       GET? Nein: weiter bei 44097
44085 JSR  57636       > CHRIN holt Zeichen vom aktiven Eingabe-Kanal
44088 STA    512       Zeichencode nach 512
44091 LDX   #255       (XR/YR) := 511
44093 LDY     #1
44095 BNE  44109       Unbedingter Sprung
```

BASIC-Routine READ (Fortsetzung)

```
44097 BMI   44216       READ? Ja: weiter bei 44216
44099 LDA     19        Eingabe ueber File?
44101 BNE   44106       Ja: weiter bei 44106
44103 JSR   43845       > "?" drucken (falls zu wenige Eingaben bei 'INPUT')
44106 JSR   44025       > Eingabe-Warteschleife
44109 STX    122        CHRGET-Pointer auf 511 setzen
44111 STY    123
44113 JSR    115        > CHRGET holt naechstes Zeichen
44116 BIT     13        String-Variable?
44118 BPL   44169       Nein: weiter bei 44169
44120 BIT     17        GET?
44122 BVC   44133       Nein: weiter bei 44133
44124 INX
44125 STX    122        CHRGET-Pointer auf 512 setzen
44127 LDA     #0
44129 STA      7        Trennzeichen
44131 BEQ   44145       Unbedingter Sprung

44133 STA      7        Code des naechsten Zeichens
44135 CMP    #34        Anfuehrungszeichen?
44137 BEQ   44146       Ja: Carry = 1, weiter bei 44146
44139 LDA    #58        Code fuer ":", Endezeichen fuer Stringuebertragung
44141 STA      7
44143 LDA    #44        Komma-Code, Endezeichen fuer Stringuebertragung
44145 CLC
44146 STA      8
44148 LDA    122        CHRGET-Pointer bei Anfuehrungszeichen um eins erhoehen
44150 LDY    123
44152 ADC     #0
44154 BCC   44157
44156 INY
44157 JSR   46221       > String uebertragen
44160 JSR   47074       > String-Ende+1 in CHRGET-Pointer bringen
44163 JSR   43482       > Stringdescriptor (Variablenwert) in Variablentabelle
44166 JMP   44177       > bis 44177 ueberspringen

44169 JSR   48371       > STRFAC, wandelt String in Gleitkommazahl um
44172 LDA     14        Flag fuer Integer
44174 JSR   43458       > FAC in Variable uebertragen
44177 JSR    121        > CHRGOT holt letztes Zeichen
44180 BEQ   44189       Trennzeichen? Ja: weiter bei 44189
44182 CMP    #44        Komma-Code?
44184 BEQ   44189       Ja: weiter bei 44189
44186 JMP   43853       > Fehlerbehandlung

44189 LDA    122        CHRGET-Pointer
44191 LDY    123
44193 STA     67        Eingabezeiger auf naechstes Datum
44195 STY     68
44197 LDA     75        CHRGET-Pointer
44199 LDY     76
44201 STA    122        wiederherstellen
44203 STY    123
44205 JSR    121        > CHRGOT holt letztes Zeichen
44208 BEQ   44255       Trennzeichen? Ja: weiter bei 44255
44210 JSR   44797       > CHKCOM prueft, ob Komma folgt
44213 JMP   44053       > Zurueck zur Eingabe, falls weitere Variablen
```

BASIC-Routine READ (Fortsetzung)

```
44216 JSR  43270       Offset zum naechsten Trennzeichen ins YR
44219 INY
44220 TAX
44221 BNE  44241       Zeilenende? Nein: weiter bei 44241
44223 LDX    #13       Code fuer "OUT OF DATA"
44225 INY
44226 LDA   (122),Y    Linkpointer high
44228 BEQ  44338       = 0? Ja: Fehlermeldung
44230 INY
44231 LDA   (122),Y    Zeilennummer low,
44233 STA     63
44235 INY
44236 LDA   (122),Y    Zeilennummer high
44238 INY
44239 STA     64       nach (63/64) bringen
44241 JSR  43259       CHRGET-Pointer um YR erhoehen
44244 JSR    121       > CHRGOT holt letztes Zeichen
44247 TAX
44248 CPX   #131       BASIC-Code fuer DATA?
44250 BNE  44216       Nein: weitersuchen ...
44252 JMP  44113       > Dateneingabe fortsetzen

44255 LDA     67       Eingabezeiger
44257 LDY     68
44259 LDX     17       INPUT/GET/READ-Flag
44261 BPL  44266       READ? Nein: weiter bei 44266
44263 JMP  43047       DATA-Pointer := Eingabezeiger

44266 LDY     #0
44268 LDA    (67),Y    Naechstes Zeichen im Puffer
44270 BEQ  44283       = Endezeichen? Ja: RTS
44272 LDA     19       zuviele Daten: Eingabe ueber File?
44274 BNE  44283       Ja: RTS
44276 LDA   #252       (Accu/YR) := 44284, Startadresse von "EXTRA IGNORED"
44278 LDY   #172
44280 JMP  43806       > Meldung drucken, RTS
44283 RTS
```

Fehlermeldungen fuer INPUT

```
44284   63  69  88  84  82  65  32                              ?EXTRA
44291   73  71  78  79  82  69  68  13   0                      IGNORED
44300   63  82  69  68  79  32                                  ?REDO
44306   70  82  79  77  32  83  84  65  82  84  13   0          FROM START
```

BASIC-Routine NEXT

```
44318 BNE  44324       folgt Variable? Ja: weiter bei 44324
44320 LDY     #0       Variablenpointer high := 0 (vgl. 41878)
44322 BEQ  44327       Unbedingter Sprung

44324 JSR  45195       > VARSUC sucht Variable
44327 STA     73       Zeiger nach (73/74)
44329 STY     74
44331 JSR  41866       > Stacksuche nach zugehoerigem FOR-NEXT-Datensatz
44334 BEQ  44341       Gefunden? Ja: weiter bei 44341
44336 LDX    #10       Code fuer "NEXT WITHOUT FOR"
44338 JMP  42039       > Fehlermeldung, READY.

44341 TXS
44342 TXA
44343 CLC
44344 ADC     #4
44346 PHA              Zeiger auf Exponenten des STEP-Werts
44347 ADC     #6
44349 STA     36       Zeiger auf Exponenten des TO-Werts
44351 PLA
44352 LDY     #1
44354 JSR  48034       > STEP-Wert nach FAC bringen
44357 TSX
44358 LDA    265,X     Vorzeichenbyte
44361 STA    102       nach (102) bringen (Vorzeichenbyte fuer FAC)
44363 LDA     73       FOR-NEXT-Variablenzeiger nach (Accu/YR)
44365 LDY     74
44367 JSR  47207       > STEP-Wert zum FOR-NEXT-Variablenwert addieren
44370 JSR  48080       > Ergebnis in FOR-NEXT-Variable uebertragen
44373 LDY     #1
44375 JSR  48221       > FOR-NEXT-Variablenwert mit TO-Wert vergleichen
44378 TSX
44379 SEC
44380 SBC    265,X
44383 BEQ  44408       Groesser: FOR-NEXT-Schleife verlassen
44385 LDA    271,X     Zeilennummer des Schleifenanfangs aus Stack
44388 STA     57       nach (57/58) bringen
44390 LDA    272,X
44393 STA     58
44395 LDA    274,X     Schleifenanfang in CHRGET-Pointer
44398 STA    122
44400 LDA    273,X
44403 STA    123
44405 JMP  42926       > Interpreterschleife

44408 TXA
44409 ADC    #17       Daten fuer FOR-NEXT-Schleife
44411 TAX              aus Stack entfernen
44412 TXS
44413 JSR    121       > CHRGOT holt letztes Zeichen
44416 CMP    #44       Komma-Code?
44418 BNE  44405       Nein: Interpreterschleife
44420 JSR    115       > CHRGET holt naechstes Zeichen
44423 JSR  44324       > Naechste FOR-NEXT-Variable bearbeiten
                         'JSR' fuer Stacksuche (vgl. 41866 ff),
                         es erfolgt kein zugehoeriges 'RTS'
```

Auswertung und Pruefung von Ausdruecken

```
44426 JSR  44446        > FRMEVL
44429 CLC               Kennzeichnung fuer Test auf numerisch
44430 BIT   ...
44431 SEC               Kennzeichnung fuer Test auf String
44432 BIT    13         Flag fuer String gesetzt?
44434 BMI  44439        Ja: weiter bei 44439
44436 BCS  44441        falls Carry gesetzt (Test auf String), TYPE MISMATCH
44438 RTS
44439 BCS  44438        falls Carry gesetzt, RTS
44441 LDX   #22         Code fuer "TYPE MISMATCH"
44443 JMP  42039        > Fehlermeldung, READY.
```

Auswertung von Ausdruecken

```
44446 LDX   122         CHRGET-Pointer
44448 BNE  44452
44450 DEC   123         um eins zuruecksetzen
44452 DEC   122
44454 LDX    #0
44456 BIT   ...
44457 PHA               Retten der Operatormaske
44458 TXA
44459 PHA               und Retten des Prioritaetswerts auf den Stack
44460 LDA    #1
44462 JSR  41979        > Falls Stackpointer < 64, dann "OUT OF MEMORY ERROR"
44465 JSR  44675        > naechstes Element des Ausdrucks auswerten
44468 LDA    #0
44470 STA    77         Operatormaske loeschen
44472 JSR   121         > CHRGOT holt letztes Zeichen
44475 SEC
44476 SBC  #177         vom Operatorcode 177 subtrahieren
44478 BCC  44503        < 0? Ja: weiter bei 44503
44480 CMP    #3         >= 3 (also Code >= 180)?
44482 BCS  44503        Ja: kein Zeichen von 177 bis 179 (>=<), weiter bei 44503
44484 CMP    #1         Hier erfolgt der Zusammenbau der Operatormaske in den
44486 ROL               Bits 0 bis 2 in (77), wenn Codes im Bereich von 177 bis
44487 EOR    #1         bis 179 folgen. Es ergeben sich folgende Moeglichkeiten:
44489 EOR    77
44491 CMP    77         Relation | > | = |>= | < |<> |<= |sonst
44493 BCC  44592          (77)   | 1 | 2 | 3 | 4 | 5 | 6 |  0
44495 STA    77
44497 JSR   115         > CHRGET holt naechstes Zeichen
44500 JMP  44475        > naechstes Zeichen der Operatormaske auswerten

44503 LDX    77         Operatormaske
44505 BNE  44551        = 0? Nein: weiter bei 44551
44507 BCS  44632        War Code >= 180? Ja: weiter bei 44632
44509 ADC    #7         War Code < 170?
44511 BCC  44632        Ja: weiter bei 44632
44513 ADC    13         Stringaddition?
44515 BNE  44520        Nein: weiter bei 44520
44517 JMP  46653        > Stringverknuepfung

44520 ADC  #255         Code wiederherstellen (Carry wurde addiert, vgl. 44513)
44522 STA    34         (34) := Code - 170 (+,-,*,/,↑,AND,OR)
44524 ASL               verdoppeln
44525 ADC    34         und (34) addieren, also mit drei multiplizieren
44527 TAY               als Zeiger ins YR
44528 PLA               Prioritaetswert der vorherigen Operation
44529 CMP  41088,Y      mit Prioritaetswert des momentanen Operators vergleichen
44532 BCS  44637        kleiner? Nein: weiter bei 44637
```

Auswertung von Ausdruecken (Fortsetzung)

```
44534 JSR  44429       > FRMNUM prueft auf numerisch
44537 PHA              Prioritaetsflag auf Stack legen
44538 JSR  44576       > Operator-Routinenadresse + Operand auf Stack, Rekursion
44541 PLA
44542 LDY    75        Operator?
44544 BPL  44569       Ja: weiter bei 44569
44546 TAX              noch offene Operationen?
44547 BEQ  44635       Nein: Exponentbyte von FAC in Accu, RTS (Auswertungsende)
44549 BNE  44646       ARG vom Stack holen

44551 LSR    13        Bit 7 im String-Flag loeschen
44553 TXA              Operatormaske
44554 ROL              um eine Stelle nach links verschieben
44555 LDX   122        CHRGET-Pointer
44557 BNE  44561
44559 DEC   123        um eins zuruecksetzen
44561 DEC   122
44563 LDY   #27        Offset von Prioritaetswert fuer >=<
44565 STA    77        als neue Operatormaske speichern
44567 BNE  44528       Unbedingter Sprung
44569 CMP  41088,Y     (Accu) mit Prioritaetsflag vergleichen
44572 BCS  44646       Groesser? Ja: weiter bei 44646
44574 BCC  44537       sonst weiter auswerten

44576 LDA  41090,Y     Adresse high der Operatorroutine
44579 PHA
44580 LDA  41089,Y     Adresse low der Operatorroutine
44583 PHA
44584 JSR  44595       > Operand auf Stack legen
44587 LDA    77        Operatormaske laden
44589 JMP  44457       und zurueck zum Routinenanfang

44592 JMP  44808       > "SYNTAX ERROR"

44595 LDA   102        Vozeichenbyte von FAC
44597 LDX  41088,Y     Prioritaetswert fuer Operation
44600 TAY
44601 PLA              Ruecksprungadresse vom Stack holen
44602 STA    34
44604 INC    34        und um eins erhoehen (nur low!)
44606 PLA
44607 STA    35
44609 TYA              Vorzeichenbyte von FAC
44610 PHA              auf Stack legen
44611 JSR  48155       > FAC runden
44614 LDA   101        FAC auf Stack ablegen
44616 PHA
44617 LDA   100
44619 PHA
44620 LDA    99
44622 PHA
44623 LDA    98
44625 PHA
44626 LDA    97
44628 PHA
44629 JMP   (34)       > Indirekter Sprung

44632 LDY  #255        Flagwert fuer fehlenden Operator
44634 PLA              Prioritaetsflag
44635 BEQ  44672       = 0? Ja: weiter bei 44672
```

Auswertung von Ausdruecken (Fortsetzung)

```
44637 CMP   #100       Prioritaetsflag fuer <=>?
44639 BEQ  44644       Ja: weiter bei 44644
44641 JSR  44429       > FRMNUM prueft auf numerisch
44644 STY    75        Flag fuer Operator
44646 PLA
44647 LSR
44648 STA    18
44650 PLA
44651 STA   105        ARG vom Stack holen
44653 PLA
44654 STA   106
44656 PLA
44657 STA   107
44659 PLA
44660 STA   108
44662 PLA
44663 STA   109
44665 PLA
44666 STA   110
44668 EOR   102        Vorzeichen von ARG mit Vorzeichen von FAC verknuepfen
44670 STA   111
44672 LDA    97        Exponentbyte von FAC
44674 RTS
```

Naechstes Element auswerten

```
44675 JMP  (778)       Normalwert des Vektors (778/779): 44678
44678 LDA    #0
44680 STA    13        String-Flag ruecksetzen
44682 JSR   115        > CHRGET holt naechstes Zeichen
44685 BCS  44690       Ziffer? Nein: weiter bei 44690
44687 JMP  48371       > Wert der folgenden Zahl nach FAC

44690 JSR  45331       > prueft, ob Buchstabe folgt
44693 BCC  44698       folgt Buchstabe?
44695 JMP  44840       Ja: weiter bei 44840

44698 CMP   #255       BASIC-Code fuer PI
44700 BNE  44717       Nein: weiter bei 44717
44702 LDA   #168       (Accu/YR) := 44712, Startadresse fuer Fliesskommazahl PI
44704 LDY   #174
44706 JSR  48034       > Konstante PI in FAC bringen
44709 JMP   115        > CHRGET holt naechstes Zeichen
```

Gleitkommakonstante PI

```
44712 130  73  15 218 161       3.14159265

44717 CMP    #46       Code fuer Dezimalpunkt?
44719 BEQ  44687       Ja: Wert der folgenden Zahl nach FAC
44721 CMP   #171       Code fuer "-"?
44723 BEQ  44813       Ja: weiter bei 44813
44725 CMP   #170       Code fuer "+"?
44727 BEQ  44682       Ja: Zeichen ignorieren, weiter bei 44682
44729 CMP    #34       Code fuer Anfuehrungszeichen?
44731 BNE  44748       Nein: weiter bei 44748
```

naechstes Element auswerten (Fortsetzung)

```
44733 LDA    122      um eins erhoehten CHRGET-Pointer
44735 LDY    123      (Carry = 1) nach (Accu/YR) bringen
44737 ADC     #0
44739 BCC  44742
44741 INY
44742 JSR  46215      > String in Stringbereich transferieren
44745 JMP  47074      > CHRGET-Pointer hinter Stringende setzen

44748 CMP   #168      BASIC-Code fuer NOT?
44750 BNE  44771      Nein: weiter bei 44771
44752 LDY    #24      Offset fuer Prioritaetswert in Tabelle
44754 BNE  44815      Unbedingter Sprung

44756 JSR  45503      > FLPINT wandelt Fliesskommazahl in 16-Bit Integer um
44759 LDA    101      Integer invertieren
44761 EOR   #255
44763 TAY
44764 LDA    100
44766 EOR   #255
44768 JMP  45969      > INTFLP wandelt Integer in (Accu/YR) in FLP in FAC um

44771 CMP   #165      BASIC-Code fuer 'FN'?
44773 BNE  44778      Nein: weiter bei 44778
44775 JMP  46068      > FN, Funktionsauswertung
44778 CMP   #180      Funktion (Interpretercodes von 180 bis 202)?
44780 BCC  44785      Nein: weiter bei 44785
44782 JMP  44967      > Berechnung von Funktionen

44785 JSR  44794      > prueft, ob "(" folgt
44788 JSR  44446      > FRMEVL wertet Klammerausdruck aus
44791 LDA    #41      Code fuer ")"
44793 BIT    ...
44794 LDA    #40      Code fuer "("
44796 BIT    ...
44797 LDA    #44      Code fuer ","
44799 LDY     #0
44801 CMP  (122),Y    mit laufendem CHRGET-Zeichen vergleichen
44803 BNE  44808      Ungleich? Ja: "SYNTAX ERROR"
44805 JMP    115      > CHRGET holt naechstes Zeichen
44808 LDX    #11      Code fuer "SYNTAX ERROR"
44810 JMP  42039      > Fehlermeldung, READY.

44813 LDY    #21      Offset auf Prioritaetswert fuer Vorzeichenwechsel
44815 PLA
44816 PLA
44817 JMP  44538      > Zurueck in Auswertungsroutine

44820 SEC
44821 LDA    100      Zeigt Variablenpointer auf Bereich von 40960 bis 58273?
44823 SBC     #0
44825 LDA    101
44827 SBC   #160
44829 BCC  44839      Nein: RTS
44831 LDA   #162
44833 SBC    100
44835 LDA   #227
44837 SBC    101
44839 RTS
```

Naechstes Element auswerten (Fortsetzung)

```
44840 JSR  45195       > VARSUC sucht Variable (kein Anlegen!, vgl. 45341 ff)
44843 STA    100       Pointer auf Variable/Descriptor nach (100/101)
44845 STY    101
44847 LDX     69       (69/70) enthaelt Namen der Variablen
44849 LDY     70
44851 LDA     13       Stringvariable?
44853 BEQ  44893       Nein: weiter bei 44893

44855 LDA     #0
44857 STA    112
44859 JSR  44820       > Zeigt Variablenpointer auf Bereich von 40960 bis 58273?
44862 BCC  44892       Nein: RTS

44864 CPX    #84       Stringvariable 'TI$'?
44866 BNE  44892
44868 CPY   #201
44870 BNE  44892       Nein: RTS

44872 JSR  44932       > Uhrzeit nach FAC bringen
44875 STY     94       (94) := 0 (Flag fuer Exponentialdarstellung
44877 DEY              von Zahlenstrings loeschen)
44878 STY    113       (113) := 255 (Pointer fuer Startadresse des Strings)
44880 LDY     #6
44882 STY     93       Anzahl der Stellen des Strings
44884 LDY    #36       Pointer auf Stellenwerte zur Berechnung von TI$
44886 JSR  48744       > String erzeugen
44889 JMP  46191       > bringt String in Stringbereich
44892 RTS

44893 BIT     14       Integer-Variable?
44895 BPL  44910       Nein: weiter bei 44910
44897 LDY     #0
44899 LDA   (100),Y    Wert der Integer-Variablen
44901 TAX
44902 INY
44903 LDA   (100),Y    aus der Variablen in FAC bringen
44905 TAY
44906 TXA
44907 JMP  45969       > INTFLP wandelt Integerzahl in Fliesskommazahl um

44910 JSR  44820       > Zeigt Variablenpointer auf Bereich von 40960 bis 58273?
44913 BCC  44960       Nein: weiter bei 44960
44915 CPX    #84       erstes Zeichen des Variablennamens = "T"?
44917 BNE  44946       Nein: weiter bei 44946
44919 CPY    #73       Variable 'TI'?
44921 BNE  44960       Nein: weiter bei 44960
44923 JSR  44932       > Uhrzeit nach FAC bringen
44926 TYA              (Accu) := 0
44927 LDX   #160       Exponentbyte fuer FAC
44929 JMP  48207       > FAC aufbereiten und linksbuendig machen

44932 JSR  65502       > RDTIM liest Uhrzeit
44935 STX    100       nach FAC uebertragen
44937 STY     99
44939 STA    101
44941 LDY     #0
44943 STY     98       (98) := 0
44945 RTS
```

Auswertung von Ausdruecken (Fortsetzung)

```
44946 CPX    #83      erstes Zeichen des Variablennamens = "S"?
44948 BNE  44960      Nein: weiter bei 44960
44950 CPY    #84      Variable 'ST'?
44952 BNE  44960      Nein: weiter bei 44960
44954 JSR  65463      > READST liest Statusvariable
44957 JMP  48188      > Statusbyte nach FAC bringen

44960 LDA    100      (Accu/YR) := Pointer auf Variable
44962 LDY    101
44964 JMP  48034      > Variable nach FAC bringen
```

Aufruf von Funktionen

```
44967 ASL             BASIC-Code verdoppeln (AND #255)
44968 PHA
44969 TAX
44970 JSR    115      > CHRGET holt naechstes Zeichen
44973 CPX   #143      BASIC-Code < 200? (143 + 256 + 1 = 200 * 2)
44975 BCC  45009      Ja: weiter bei 45009
```

'LEFT$', 'RIGHT$', 'MID$'

```
44977 JSR  44794      > prueft, ob "(" folgt
44980 JSR  44446      > FRMEVL wertet Ausdruck aus
44983 JSR  44797      > CHKCOM prueft, ob Komma folgt
44986 JSR  44431      > FRMNUM prueft, ob Stringvariable
44989 PLA
44990 TAX
44991 LDA    101      Adresse des Stringdescriptors
44993 PHA
44994 LDA    100
44996 PHA
44997 TXA             BASIC-Code * 2 (AND #255) auf Stack retten
44998 PHA
44999 JSR  47006      > GETBYT bringt zweiten Parameter ins XR
45002 PLA
45003 TAY             BASIC-Code fuer Funktionsroutinenaufruf ins YR
45004 TXA
45005 PHA             zweites Argument auf Stack
45006 JMP  45014      > Funktionsroutinenaufruf

45009 JSR  44785      > wertet Klammerausdruck aus
45012 PLA             BASIC-Code fuer Funktionsroutinenaufruf ins YR
45013 TAY
45014 LDA  40938,Y    Startadresse der Funktion nach (85/86)
45017 STA     85      bringen (in 84 steht der Code fuer JMP)
45019 LDA  40939,Y
45022 STA     86
45024 JSR     84      > Funktionsberechnung
                        (Rueckkehr nur bei Funktionen mit numerischen Resultat)
45027 JMP  44429      > FRMNUM prueft, ob numerisches Ergebnis
```

Ausfuehrung von AND und OR

```
45030 LDY   #255      Einsprung fuer 'OR'
45032 BIT    ...
45033 LDY     #0      Einsprung fuer 'AND'
45035 STY     11      Flag fuer 'AND' oder 'OR' setzen
45037 JSR  45503      > FLPINT wandelt FAC in 16-Bit Integer nach (100/101)
```

Ausfuehrung von AND und OR (Fortsetzung)

```
45040 LDA    100
45042 EOR     11
45044 STA      7
45046 LDA    101
45048 EOR     11
45050 STA      8
45052 JSR  48124       > FAC := ARG
45055 JSR  45503       > FLPINT wandelt FAC in 16-Bit Integer nach (100/101)
45058 LDA    101
45060 EOR     11
45062 AND      8
45064 EOR     11
45066 TAY
45067 LDA    100
45069 EOR     11
45071 AND      7
45073 EOR     11
45075 JMP  45969       > INTFLP wandelt Integer in FLP in FAC mit Vorzeichen
```

Ausfuehrung von <=>

```
45078 JSR  44432       > Prueft, ob richtiger Variablentyp
45081 BCS  45102       String? Ja: weiter bei 45102
45083 LDA    110       ARG in Speicherformat umwandeln
45085 ORA   #127
45087 AND    106
45089 STA    106
45091 LDA   #105       (Accu/YR) := 105, Adresse von ARG
45093 LDY     #0
45095 JSR  48219       > vergleicht FAC mit ARG
45098 TAX
45099 JMP  45153       > bringt Wahrheitswert (0 oder -1) nach FAC

45102 LDA     #0
45104 STA     13       String-Flag zuruecksetzen
45106 DEC     77       Operator-Maske
45108 JSR  46758       > FRESTR
45111 STA     97       (97) := Stringlaenge
45113 STX     98       (98/99) := Startadresse des ersten Strings
45115 STY     99
45117 LDA    108       Zeiger auf zweiten String
45119 LDY    109
45121 JSR  46762       > FRESTR (mit Descriptorpointer in (Accu/YR))
45124 STX    108       Startadresse des zweiten Strings nach (108/109)
45126 STY    109
45128 TAX              Laenge des zweiten Strings im XR merken
45129 SEC
45130 SBC     97
45132 BEQ  45142       Zweiter String genauso lang wie erster: weiter bei 45142
45134 LDA     #1       Flagwert fuer 'erster String laenger'
45136 BCC  45142       zweiter String kuerzer: weiter bei 45142
45138 LDX     97       Laenge des ersten Strings
45140 LDA   #255       Flagwert fuer 'erster String kuerzer'
45142 STA    102       Flag fuer Wahrheitswert, falls Strings bis zum letzen
45144 LDY   #255       Zeichen des kuerzeren Strings identisch, jedoch
45146 INX              verschieden lang sind.
45147 INY
45148 DEX
45149 BNE  45158       Beide Strings Zeichen fuer Zeichen vergleichen
45151 LDX    102
45153 BMI  45170
```

Ausfuehrung von <=>, Vergleich zweier Strings (Fortsetzung)

```
45155 CLC
45156 BCC  45170
45158 LDA   (108),Y   beide Strings zeichenweise vergleichen
45160 CMP    (98),Y
45162 BEQ  45147      Gleich? Ja: weitermachen ...
45164 LDX   #255      Vergleich abschliessen
45166 BCS  45170
45168 LDX     #1
45170 INX
45171 TXA
45172 ROL
45173 AND     18
45175 BEQ  45179
45177 LDA   #255
45179 JMP  48188      > Wahrheitswert (0 oder -1) nach FAC bringen
```

Verwaltung der Variablen

```
45182 JSR  44797      > CHKCOM prueft auf Komma
```

Einsprung fuer BASIC-Routine DIM

```
45185 TAX             naechstes Zeichen
45186 JSR  45200      > Feld dimensionieren (Name und Groesse werden als
                        Variable gelesen und entsprechend dimensioniert)
45189 JSR    121      > CHRGOT holt letztes Zeichen
45192 BNE  45182      weitere Dimensionierung? Ja: zurueck nach 45182
45194 RTS
```

VARSUC, sucht Variable oder legt sie an

```
45195 LDX     #0      Flagwert fuer fehlende Dimensionierung
45197 JSR    121      > CHRGOT holt letztes Zeichen
45200 STX     12      Flag fuer DIM
45202 STA     69      Variablenname, erstes Zeichen
45204 JSR    121      > CHRGOT holt letztes Zeichen
45207 JSR  45331      > prueft, ob Buchstabe
45210 BCS  45215      Buchstabe? Ja: weitermachen
45212 JMP  44808      > "SYNTAX ERROR"
45215 LDX     #0
45217 STX     13      String-Flag ruecksetzen
45219 STX     14      Integer-Flag ruecksetzen
45221 JSR    115      > CHRGET holt naechstes Zeichen
45224 BCC  45231      Ziffer? Ja: weiter bei 45231
45226 JSR  45331      > prueft, ob Buchstabe
45229 BCC  45242      Buchstabe? Nein: weiter bei 45242
45231 TAX             zweites Zeichen des Variablennnamens im XR merken
45232 JSR    115      > CHRGET holt naechstes Zeichen
45235 BCC  45232      Ziffer? Ja: weitere Zeichen lesen
45237 JSR  45331      > prueft, ob Buchstabe
45240 BCS  45232      Buchstabe? Ja: weiter Zeichen lesen
45242 CMP    #36      Code fuer "$"?
45244 BNE  45252      Nein: weiter bei 45252
45246 LDA   #255
45248 STA     13      String-Flag setzen
45250 BNE  45268      Unbedingter Sprung
45252 CMP    #37      Code fuer "%"
45254 BNE  45275      Nein: weiter bei 45275
45256 LDA     16      Integer erlaubt (zum Beispiel als FOR-NEXT-Variable)?
45258 BNE  45212      Nein: "SYNTAX ERROR"
```

Verwaltung der Variablen (Fortsetzung)

```
45260 LDA   #128
45262 STA    14       Integer-Flag setzen
45264 ORA    69       zur Kennzeichnung als Integer-Variable Bit 7 im ersten
45266 STA    69       und zweiten Zeichen des Variablennamens setzen
45268 TXA
45269 ORA   #128      Bit 7 im zweiten Zeichen des Variablennamens setzen
45271 TAX
45272 JSR   115       > CHRGET holt naechstes Zeichen
45275 STX    70       Variablenname (zweites Zeichen) abspeichern
45277 SEC             Annahme von Feldvariablen erlaubt?
45278 ORA    16       Wenn nicht, dann Bit 7 setzen (Accu kann nicht = 40 sein)
45280 SBC   #40       Code fuer "(" subtrahieren
45282 BNE  45287      = 0? Nein: weiter bei 45287
45284 JMP  45521      > Bearbeitung von Feldvariablen

45287 LDY    #0
45289 STY    16       (16) := 0
45291 LDA    45       Variablen-Anfangspointer
45293 LDX    46
45295 STX    96       als Suchpointer speichern
45297 STA    95
45299 CPX    48       Suchpointer = Feldvariablen-Anfangspointer?
45301 BNE  45307      Nein: weiter bei 45307
45303 CMP    47
45305 BEQ  45341      Ja: weiter bei 45341
45307 LDA    69       Variablenname, erstes Zeichen
45309 CMP   (95),Y    mit Eintrag vergleichen
45311 BNE  45321      Ungleich: weitersuchen ...
45313 LDA    70       Variablenname, zweites Zeichen
45315 INY
45316 CMP   (95),Y    mit Eintrag vergleichen
45318 BEQ  45445      Gleich: Variable gefunden, weiter bei 45445
45320 DEY
45321 CLC
45322 LDA    95       Suchzeiger
45324 ADC    #7       um 7 erhoehen (Laenge eines Variableneintrags)
45326 BCC  45297
45328 INX             Uebertrag addieren
45329 BNE  45295      als neuen Suchzeiger speichern, zurueck zum Anfang

45331 CMP   #65       Pruefung, ob Accu einen Buchstabencode enthaelt
45333 BCC  45340      Wenn ja, so ist die Carry-Flag gesetzt,
45335 SBC   #91       ansonsten geloescht
45337 SEC
45338 SBC  #165
45340 RTS

45341 PLA             Ruecksprungadresse low der aufrufenden Routine
45342 PHA
45343 CMP   #42       Adresse low bei Aufruf von 44840
45345 BNE  45352      Nein: weiter bei 45352
45347 LDA   #19       (Accu/YR) := 48915, Adresse von Fliesskommazahl 0
45349 LDY  #191       als Variablenpointer fuer nichtdefinierte Variablen
45351 RTS             und Systemvariable TI$ (vgl. 43482 ff)

45352 LDA    69       Variablenname
45354 LDY    70
45356 CMP   #84       erstes Zeichen des Namens = 'T'?
45358 BNE  45371      Nein: weiter bei 45371
45360 CPY  #201       Variable 'TI$'?
45362 BEQ  45347      Ja: weiter bei 45347
```

Verwaltung der Variablen (Fortsetzung)

```
45364 CPY   #73      Variable 'TI'?
45366 BNE  45371     Nein: weiter bei 45371
45368 JMP  44808     > "SYNTAX ERROR" (Zuweisung an TI nicht erlaubt)

45371 CMP   #83      erstes Zeichen des Namens = "S"?
45373 BNE  45379     Nein: weiter bei 45379
45375 CPY   #84      Statusvariable 'ST'?
45377 BEQ  45368     Ja: "SYNTAX ERROR" (Zuweisung an ST nicht erlaubt)

45379 LDA    47      Feldvariablen-Anfangspointer
45381 LDY    48
45383 STA    95      Startadresse fuer Transfer (Quellbereich)
45385 STY    96
45387 LDA    49      Variablen-Endpointer+1 (Quellbereich)
45389 LDY    50
45391 STA    90      Endadresse+1 fuer Transfer
45393 STY    91
45395 CLC            Tabelle der Arrays muss fuer Variablenzuweisung
45396 ADC    #7      um sieben Bytes nach oben verschoben werden
45398 BCC  45401
45400 INY            Uebertrag addieren
45401 STA    88      Endadresse+1 fuer Zielbereich
45403 STY    89
45405 JSR  41912     > Block-Verschiebe-Routine
45408 LDA    88
45410 LDY    89
45412 INY
45413 STA    47      Neuer Feldvariablen Startpointer
45415 STY    48

45417 LDY    #0      Neue Variable einbauen
45419 LDA    69      Variablenname und Variablentyp
45421 STA   (95),Y   als erste zwei Bytes in
45423 INY            Variable ablegen
45424 LDA    70
45426 STA   (95),Y
45428 LDA    #0      Rest des Variablenwerts mit Nullen auffuellen
45430 INY
45431 STA   (95),Y
45433 INY
45434 STA   (95),Y
45436 INY
45437 STA   (95),Y
45439 INY
45440 STA   (95),Y
45442 INY
45443 STA   (95),Y

45445 LDA    95      (Accu/YR) zeigt auf erstes Byte des Variablenwerts
45447 CLC
45448 ADC    #2      um zwei erhoehen
45450 LDY    96
45452 BCC  45455
45454 INY            zeigt nun auf erstes Byte des Variablenwerts
45455 STA    71      als Variablenzeiger nach (71/72) bringen
45457 STY    72
45459 RTS
```

Laenge Arrayheader zu (95/96) addieren, ergibt Zeiger auf erstes Feldelement

```
45460 LDA     11       Anzahl der Dimensionen
45462 ASL              verdoppeln (2 Bytes je Dimension)
45463 ADC     #5       plus fuenf Bytes fuer Arrayheader
45465 ADC     95
45467 LDY     96
45469 BCC  45472
45471 INY              Uebertrag addieren
45472 STA     88       Ergebnis nach (88/89)
45474 STY     89
45476 RTS

45477 144 128   0   0   0     Gleitkommadarstellung fuer -32768

45482 JSR  45503       > FLPINT
45485 LDA    100
45487 LDY    101
45489 RTS

45490 JSR    115       > CHRGET holt naechstes Zeichen
45493 JSR  44446       > FRMEVL wertet Ausdruck auf, Ergebnis in FAC
45496 JSR  44429       > FRMNUM prueft, ob numerische Variable
45499 LDA    102       Vorzeichenbyte von FAC
45501 BMI  45516       Bit 7 gesetzt? Ja: "ILLEGAL QUANTITY ERROR"
```

FLPINT

```
45503 LDA     97       Exponentbyte von FAC
45505 CMP   #144       Exponent < 16?
45507 BCC  45518       Ja: weiter bei 45518
45509 LDA   #165       (Accu/YR) := 45477, Startadresse fuer Konstante -32768
45511 LDY   #177
45513 JSR  48219       > Vergleich zwischen FAC und Konstante
45516 BNE  45640       Ungleich? Ja: "ILLEGAL QUANTITY ERROR"
45518 JMP  48283       > Fliesskomma in Integer-Format bringen
```

Verwaltung von Feldvariablen

```
45521 LDA     12       Flag fuer DIM
45523 ORA     14       Integer-Flag (Bit 7 gesetzt, falls Integer),
45525 PHA
45526 LDA     13       Flag fuer String/numerisch auf Stack retten
45528 PHA
45529 LDY     #0       Zaehler fuer Anzahl Dimensionen,
45531 TYA
45532 PHA
45533 LDA     70       Variablenname, zweites Zeichen,
45535 PHA
45536 LDA     69       Variablenname, erstes Zeichen auf Stack retten
45538 PHA
45539 JSR  45490       > Index-Auswertung (16-Bit Integer lesen)
45542 PLA
45543 STA     69       Variablenname, zweites Zeichen,
45545 PLA
45546 STA     70       Variablenname, erstes Zeichen,
45548 PLA
45549 TAY              Zaehler fuer Anzahl Dimensionen vom Stack holen
45550 TSX
45551 LDA    258,X     Variablenflags aus dem Stack kopieren
45554 PHA              und wieder oben auf den Stack legen
45555 LDA    257,X
45558 PHA
```

Verwaltung von Feldvariablen (Fortsetzung)

```
45559 LDA    100        Indexbytes an Stelle der Variablenflags
45561 STA    258,X      auf den Stack legen
45564 LDA    101
45566 STA    257,X
45569 INY               Dimensionszaehler erhoehen
45570 JSR    121        > CHRGOT holt letztes Zeichen
45573 CMP    #44        Komma (weiterer Index)?
45575 BEQ   45531       Ja: zurueck nach 45531
45577 STY    11         Anzahl der Dimensionen merken
45579 JSR   44791       > prueft, ob ")" folgt
45582 PLA
45583 STA    13         Flag fuer String/numerisch,
45585 PLA
45586 STA    14         Flag fuer Integer wiederherstellen
45588 AND   #127
45590 STA    12
45592 LDX    47         Feldvariablen-Anfangspointer
45594 LDA    48
45596 STX    95         als Suchpointer initialisieren
45598 STA    96
45600 CMP    50         Ende der Feldvariablen erreicht?
45602 BNE   45608       Nein: weitersuchen ...
45604 CPX    49
45606 BEQ   45665       Ja: Variable nicht gefunden, weiter bei 45665
45608 LDY    #0
45610 LDA    (95),Y     Variablenname auf Arrayheader
45612 INY
45613 CMP    69         mit gesuchtem Variablennamen vergleichen
45615 BNE   45623       Ungleich: weiter bei 45623
45617 LDA    70         zweites Zeichen des gesuchten Variablennamens
45619 CMP    (95),Y     mit Feldvariablenname vergleichen
45621 BEQ   45645       Gleich: Feld gefunden, weiter bei 45645
45623 INY
45624 LDA    (95),Y     sonst Feldlaenge zum Suchzeiger addieren,
45626 CLC               ergibt Pointer auf naechstes Array
45627 ADC    95
45629 TAX
45630 INY
45631 LDA    (95),Y
45633 ADC    96
45635 BCC   45596
45637 LDX    #18        Code fuer "BAD SUBSCRIPT"
45639 BIT    ...
45640 LDX    #14        Code fuer "ILLEGAL QUANTITY"
45642 JMP   42039       > Fehlermeldung, READY.

45645 LDX    #19        Code fuer "REDIM'D ARRAY"
45647 LDA    12         DIM-Flag = 0 (Variable gefunden)?
45649 BNE   45642       Nein: "REDIM'D ARRAY ERROR"
45651 JSR   45460       > (88/89) als Pointer auf erste Feldvariable
45654 LDA    11         Dimensionszaehler
45656 LDY    #4
45658 CMP    (95),Y     mit Anzahl an Dimensionen im Arrayheader vergleichen
45660 BNE   45637       Ungleich: "BAD SUBSCRIPT ERROR"
45662 JMP   45802       > Suche nach richtigem Element des Arrays

45665 JSR   45460       > Laenge des Variablenkopfs zum Suchzeiger addieren
45668 JSR   41992       > prueft, ob genug Platz im Speicher
```

Verwaltung der Feldvariablen (Fortsetzung)

```
45671 LDY     #0
45673 STY    114
45675 LDX     #5      Laenge eines Elements im Array bei Real
45677 LDA     69      Variablenname, erstes Zeichen
45679 STA    (95),Y   in Arrayheader bringen
45681 BPL  45684      Integer? Nein: weiter bei 45684
45683 DEX             bei Integer Anzahl Bytes je Element vermindern
45684 INY
45685 LDA     70      Variablenname, zweites Zeichen
45687 STA    (95),Y   in Arrayheader bringen
45689 BPL  45693      Fliesskomma? Ja: weiter 45693
45691 DEX             sonst Anzahl Bytes je Element um zwei vermindern
45692 DEX
45693 STX    113      Elementlaenge (FLP: 5, String: 3, Integer: 2)
45695 LDA     11      Anzahl der Dimensionen
45697 INY
45698 INY
45699 INY
45700 STA    (95),Y   in Arrayheader bringen
45702 LDX    #11      (XR/Accu) := 11, Ersatzwert bei fehlendem DIM
45704 LDA     #0
45706 BIT     12      Aufruf durch 'DIM'?
45708 BVC  45718      Nein: weiter bei 45718
45710 PLA             Werte der Dimensionierung vom Stack holen
45711 CLC
45712 ADC     #1      um eins erhoehen (Feld mit Index 0)
45714 TAX
45715 PLA
45716 ADC     #0      Uebertrag addieren
45718 INY
45719 STA    (95),Y   DIM-Wert in Arrayheader bringen
45721 INY
45722 TXA
45723 STA    (95),Y
45725 JSR  45900      > benoetigten Speicherplatz fuer Dimension(en) berechnen
45728 STX    113      Speicherbedarf low
45730 STA    114      Speicherbedarf high
45732 LDY     34      Zeiger auf Arrayheader
45734 DEC     11      weitere Dimensionen?
45736 BNE  45702      Ja: zurueck zum Schleifenbeginn
45738 ADC     89      Laenge des Feldes zur Anfangsadresse addieren
45740 BCS  45835      16-Bit-Ueberlauf? Ja: "OUT OF MEMORY ERROR"
45742 STA     89
45744 TAY
45745 TXA
45746 ADC     88
45748 BCC  45753
45750 INY             (Accu/YR) := Endadresse des Feldes
45751 BEQ  45835      16-Bit-Ueberlauf? Ja: "OUT OF MEMORY ERROR"
45753 JSR  41992      > prueft, ob genug Speicherplatz
45756 STA     49      neuen Endezeiger fuer Variablen setzen
45758 STY     50
45760 LDA     #0      Alle Elemente des Feldes mit Nullen vorbesetzen
45762 INC    114
45764 LDY    113
45766 BEQ  45773
45768 DEY
45769 STA    (88),Y
45771 BNE  45768
45773 DEC     89
45775 DEC    114
```

Verwaltung der Feldvariablen (Fortsetzung)

```
45777 BNE  45768
45779 INC     89
45781 SEC
45782 LDA     49      Endezeiger des Feldes minus
45784 SBC     95      Zeiger auf Arrayheader ergibt Laenge des Feldes
45786 LDY     #2
45788 STA   (95),Y    in dritte und vierte Position des Arrayheaders
45790 LDA     50      schreiben
45792 INY
45793 SBC     96
45795 STA   (95),Y
45797 LDA     12      Aufruf durch 'DIM'-Befehl?
45799 BNE  45899      Ja: RTS
45801 INY             ansonsten Feldelement suchen
45802 LDA   (95),Y    Anzahl an Dimensionen
45804 STA     11
45806 LDA     #0
45808 STA    113
45810 STA    114
45812 INY
45813 PLA             Feldindex vom Stack holen
45814 TAX
45815 STA    100      nach (100/101) bringen
45817 PLA
45818 STA    101
45820 CMP   (95),Y    mit durch 'DIM' gesetzten Grenzen vergleichen
45822 BCC  45838      Kleiner? Ja: weiter bei 45838
45824 BNE  45832      sonst: "BAD SUBSCRIPT ERROR"
45826 INY
45827 TXA             ebenso im Falle von Gleichheit der Bytes high die
45828 CMP   (95),Y    Bytes low ueberpruefen
45830 BCC  45839      kleiner? Ja: weiter bei 45839
45832 JMP  45637      > "BAD SUBSCRIPT ERROR"

45835 JMP  42037      > "OUT OF MEMORY ERROR"

45838 INY             Berechnung der Position eines Feldelements
45839 LDA    114      innerhalb eines Arrays
45841 ORA    113
45843 CLC
45844 BEQ  45856
45846 JSR  45900      > (XR/YR) := (113/114)*((95/96),Y)
45849 TXA
45850 ADC    100
45852 TAX
45853 TYA
45854 LDY     34      Zeiger in Arrayheader
45856 ADC    101
45858 STX    113
45860 DEC     11      Zaehler fuer Anzahl Dimensionen vermindern
45862 BNE  45810
45864 STA    114
45866 LDX     #5      Laenge eines Elements (Real-Array)
45868 LDA     69      Bit 7 im ersten Zeichen des Namens gesetzt (Integer)?
45870 BPL  45873      Nein: weiter bei 45873
45872 DEX             Elementlaenge vermindern
45873 LDA     70      Bit 7 im zweiten Zeichen des Namens gesetzt (FLP)?
45875 BPL  45879      Ja: weiter bei 45879
45877 DEX
45878 DEX
45879 STX     40      Laenge eines Feldelements
```

Verwaltung der Variablen (Fortsetzung)

```
45881 LDA      #0
45883 JSR   45909      > Berechnung des Offsets des Feldelements
45886 TXA              zur Adresse des ersten Feldelements addieren
45887 ADC      88      ergibt Position des gesuchten Elements
45889 STA      71
45891 TYA
45892 ADC      89
45894 STA      72      Adresse in Variablenpointer (71/72) und (Accu/YR) bringen
45896 TAY
45897 LDA      71
45899 RTS
```

16-Bit-Binaermultiplikation von (40/41) mit (113/114), Ergebnis nach (XR/YR)

```
45900 STY      34      YR merken
45902 LDA    (95),Y    (40/41) := ((95/96),Y)
45904 STA      40
45906 DEY
45907 LDA    (95),Y
45909 STA      41
45911 LDA     #16      Bit-Verschiebezaehler
45913 STA      93
45915 LDX      #0      Bytes low und high des Ergebnisregisters (XR/YR)
45917 LDY      #0      mit Null vorbesetzen
45919 TXA              (XR/YR) fuer naechsten Schritt der
45920 ASL              Binaermultiplikation verdoppeln
45921 TAX
45922 TYA
45923 ROL
45924 TAY              Ueberlauf in Bit 16?
45925 BCS   45835      Ja: "OUT OF MEMORY ERROR"
45927 ASL     113      naechstes Bit aus (113/114) herausshiften
45929 ROL     114
45931 BCC   45944      Carry = 0? Ja: Addition uebergehen
45933 CLC              (XR/YR) := (XR/YR) + (40/41)
45934 TXA
45935 ADC      40
45937 TAX
45938 TYA
45939 ADC      41
45941 TAY
45942 BCS   45835      Ueberlauf bei Addition? Ja: "OUT OF MEMORY ERROR"
45944 DEC      93      naechstes Bit herausshiften
45946 BNE   45919      alle 16 Bits verschoben? Nein: weitermachen ...
45948 RTS
```

BASIC-Funktion FRE

```
45949 LDA      13      String-Flag gesetzt?
45951 BEQ   45956      Nein: weiter bei 45956
45953 JSR   46758      > FRESTR
45956 JSR   46374      > GARBAGE COLLECT
45959 SEC
45960 LDA      51      Vom String-Anfangspointer den
45962 SBC      49      Variablen-Endezeiger subtrahieren,
45964 TAY              daraus ergibt sich der freie Speicherplatz
45965 LDA      52
45967 SBC      50
```

INFLP: Umwandlung einer 16-Bit-Integer-Zahl mit Vorzeichen in Gleitkomma

```
45969 LDX     #0
45971 STX     13       Flag fuer String loeschen
45973 STA     98       Integer in (YR/Accu) nach (99/98) bringen
45975 STY     99
45977 LDX   #144       Exponent := 16 (144 - 128)
45979 JMP  48196       > Fliesskommazahl in FAC erzeugen
```

BASIC-Funktion POS

```
45982 SEC              Flagwert fuer 'Cursorposition lesen'
45983 JSR  65520       > PLOT, Cursorposition lesen, Spalte ins YR
45986 LDA     #0       Byte high loeschen
45988 BEQ  45969       Unbedingter Sprung nach INTFLP
```

Test auf Direktmodus ("ILLEGAL DIRECT ERROR")

```
45990 LDX     58       Zeilennummer high (Im Direktmodus = 255)
45992 INX
45993 BNE  45899       Direktmodus? Nein: RTS
45995 LDX    #21       Code fuer "ILLEGAL DIRECT"
45997 BIT    ...
45998 LDX    #27       Code fuer "UNDEF'D FUNCTION"
46000 JMP  42039       > Fehlermeldung, READY.
```

BASIC-Routine DEF

```
46003 JSR  46049       > FN-Namen pruefen, FN-Pointer setzen (78/79)
46006 JSR  45990       > prueft auf Direktmodus
46009 JSR  44794       > prueft, ob "(" folgt
46012 LDA   #128
46014 STA     16       Annahme von Integer- und Feldvariablen sperren
46016 JSR  45195       > VARSUC sucht Variable oder legt sie an
46019 JSR  44429       > FRMNUM prueft, ob numerische Variable
46022 JSR  44791       > prueft, ob ")" folgt
46025 LDA   #178       BASIC-Code fuer "="
46027 JSR  44799       > SYNCHR prueft, ob dieser Code folgt
46030 PHA              erstes Zeichen des Ausdrucks auf Stack legen (unwichtig)
46031 LDA     72       ebenso den Pointer auf die FN-Variable
46033 PHA
46034 LDA     71
46036 PHA
46037 LDA    123       CHRGET-Pointer (zeigt auf FN-Ausdruck)
46039 PHA              auf Stack legen
46040 LDA    122
46042 PHA
46043 JSR  43256       > DATA, setzt CHRGET-Pointer auf naechstes Trennzeichen
46046 JMP  46159       > Bytes vom Stack in FN-Namensvariable bringen

46049 LDA   #165       BASIC-Code fuer 'FN'
46051 JSR  44799       > SYNCHR prueft, ob dieser Code folgt
46054 ORA   #128
46056 STA     16       Annahme von Integer- und Feldvariablen sperren
46058 JSR  45202       > VARSUC sucht Variable oder legt sie an
46061 STA     78       FN-Pointer auf Variable setzen
46063 STY     79
46065 JMP  44429       > FRMNUM prueft, ob numerische Variable
```

BASIC-Funktion FN

```
46068 JSR  46049       > FN-Namen pruefen, FN-Pointer setzen (78/79)
46071 LDA    79        Pointer fuer FN-Namen auf Stack legen
46073 PHA
46074 LDA    78
46076 PHA
46077 JSR  44785       > Klammern pruefen, Ausdruck in Klammern auswerten
46080 JSR  44429       > FRMNUM prueft, ob numerischer Ausdruck
46083 PLA
46084 STA    78
46086 PLA
46087 STA    79
46089 LDY    #2
46091 LDA   (78),Y     Zeiger low auf FN-Variable
46093 STA    71
46095 TAX
46096 INY
46097 LDA   (78),Y     Zeiger high auf FN-Variable = 0 (FN-Variable wurde erst
46099 BEQ  45998       durch FN-Aufruf angelegt? Ja: "UNDEF'D FUNCTION ERROR"
46101 STA    72
46103 INY
46104 LDA   (71),Y     Wert der FN-Variablen auf Stack retten
46106 PHA
46107 DEY
46108 BPL  46104
46110 LDY    72
46112 JSR  48084       > FAC runden und in FN-Variable bringen
46115 LDA   123        CHRGET-Pointer auf Stack retten
46117 PHA
46118 LDA   122
46120 PHA
46121 LDA   (78),Y     CHRGET-Pointer auf Funktionsausdruck setzen
46123 STA   122
46125 INY
46126 LDA   (78),Y
46128 STA   123
46130 LDA    72        Pointer auf FN-Variable auf Stack retten
46132 PHA
46133 LDA    71
46135 PHA
46136 JSR  44426       > FRMEVL, FRMNUM, numerischen Ausdruck auswerten
46139 PLA              Pointer auf FN-Variable nach (78/79)
46140 STA    78
46142 PLA
46143 STA    79
46145 JSR   121        > CHRGOT holt letztes Zeichen
46148 BEQ  46153       Trennzeichen? Ja: weiter bei 46153
46150 JMP  44808       > "SYNTAX ERROR"
```

Abschluss der FN-Auswertung

```
46153 PLA              CHRGET-Pointer wiederherstellen
46154 STA   122
46156 PLA
46157 STA   123
46159 LDY    #0        alten Wert der FN-Variablen wiederherstellen
46161 PLA
46162 STA   (78),Y
46164 PLA
46165 INY
46166 STA   (78),Y
46168 PLA
```

BASIC-Funktion FN (Fortsetzung)

```
46169 INY
46170 STA    (78),Y
46172 PLA
46173 INY
46174 STA    (78),Y
46176 PLA
46177 INY
46178 STA    (78),Y
46180 RTS
```

BASIC-Funktion STR$

```
46181 JSR  44429       > FRMNUM prueft, ob numerische Variable
46184 LDY     #0
46186 JSR  48607       > FACSTR erzeugt String aus FAC ab (255)
46189 PLA              Ruecksprungadresse in Funktionsauswertung
46190 PLA              vom Stack entfernen
46191 LDA   #255       (Accu/YR) := 255, Startadresse des Strings
46193 LDY     #0
46195 BEQ  46215       Unbedingter Sprung
```

Speicherplatz pruefen, Stringpointer setzen

```
46197 LDX    100       (100/101) zeigt auf Stringdescriptor
46199 LDY    101
46201 STX     80
46203 STY     81
46205 JSR  46324       > Speicherplatz pruefen, String-Anfangspointer setzen
46208 STX     98       Startadresse des Strings nach (98/99)
46210 STY     99
46212 STA     97       Laenge nach (97)
46214 RTS
```

String lesen und in Stringbereich transferieren

```
46215 LDX    #34       Code fuer Anfuehrungszeichen nach (7) und (8)
46217 STX      7
46219 STX      8
46221 STA    111       Startadresse des Strings nach (111/112) und (98/99)
46223 STY    112
46225 STA     98
46227 STY     99
46229 LDY   #255       Pointer auf String initialisieren
46231 INY
46232 LDA  (111),Y     Endezeichen?
46234 BEQ  46248       Ja: weiter bei 46248
46236 CMP      7
46238 BEQ  46244
46240 CMP      8
46242 BNE  46231
46244 CMP    #34
46246 BEQ  46249
46248 CLC
46249 STY     97       Stringlaenge
46251 TYA
46252 ADC    111       zur Anfangsadresse des Strings addieren
46254 STA    113       ergibt Endadresse+1 des Strings
46256 LDX    112       Uebertrag addieren
46258 BCC  46261
46260 INX
46261 STX    114
```

String lesen und in Stringbereich transferieren (Fortsetzung)

```
46263 LDA    112       Startadresse high
46265 BEQ  46271       = 0? Ja: weiter bei 46271
46267 CMP     #2       = 2?
46269 BNE  46282       Nein: weiter bei 46282
46271 TYA              Stringlaenge
46272 JSR  46197       > Speicherplatz pruefen, Stringpointer setzen
46275 LDX    111       Anfangsadresse des Strings in (111/112)
46277 LDY    112
46279 JSR  46728       > String in oberen Stringbereich transferieren
46282 LDX     22       Descriptor-Index fuer Tabelle (25,...,33)
46284 CPX    #34       Tabelle voll?
46286 BNE  46293       Nein: weiter bei 46293
46288 LDX    #25       Code fuer "FORMULA TOO COMPLEX"
46290 JMP  42039       > Fehlermeldung, READY.
46293 LDA     97       Stringdescriptor (Laenge; Startadresse low, high)
46295 STA     0,X      in Descriptortabelle bringen
46297 LDA     98
46299 STA     1,X
46301 LDA     99
46303 STA     2,X
46305 LDY     #0       (XR/YR) := Pointer auf Descriptor in Tabelle (25,...,33)
46307 STX    100       (100/101) := (XR/YR) (Pointer auf Descriptor)
46309 STY    101
46311 STY    112       (112) := 0
46313 DEY
46314 STY     13       Stringflag setzen
46316 STX     23       Zeiger auf letzten Descriptor setzen
46318 INX
46319 INX
46320 INX
46321 STX     22       Descriptor-Index um drei erhoehen
46323 RTS

46324 LSR     15       Flag fuer Garbage Collect ruecksetzen
46326 PHA              Stringlaenge retten
46327 EOR   #255       String-Anfangspointer (51/52)
46329 SEC              um Stringlaenge vermindern
46330 ADC     51
46332 LDY     52
46334 BCS  46337
46336 DEY
46337 CPY     50       Ist String-Anfangzeiger kleiner als Zeiger
46339 BCC  46358       auf Ende der Variablentabelle? Ja: weiter bei 46358
46341 BNE  46347
46343 CMP     49
46345 BCC  46358
46347 STA     51       Zeiger als neuen Stringanfangspointer setzen
46349 STY     52
46351 STA     53
46353 STY     54
46355 TAX
46356 PLA              Stringlaenge wieder vom Stack holen
46357 RTS
46358 LDX    #16       Code fuer "OUT OF MEMORY"
46360 LDA     15       Garbage Collect bereits ausgefuehrt?
46362 BMI  46290       Ja: "OUT OF MEMORY ERROR"
46364 JSR  46374       > GARBAGE COLLECT
46367 LDA   #128
46369 STA     15       Flag fuer Garbage Collect setzen
46371 PLA              Stringlaenge wieder vom Stack holen
46372 BNE  46326       Nochmals Einbau des Strings versuchen
```

GARBAGE COLLECT, Stringmuellbeseitigung

```
46374 LDX      55       Pointer auf Ende Arbeitsspeicher
46376 LDA      56
46378 STX      51       als String-Anfangspointer setzen
46380 STA      52
46382 LDY      #0
46384 STY      79
46386 STY      78
46388 LDA      49       Variablen-Endepointer
46390 LDX      50
46392 STA      95       nach (95/96) bringen
46394 STX      96
46396 LDA     #25       Startadresse der Descriptorentabelle (25,...,33)
46398 LDX      #0
46400 STA      34       als Suchpointer nach (34/35) bringen
46402 STX      35
46404 CMP      22       identisch mit Pointer auf letzten String
46406 BEQ   46413       Ja: weiter bei 46413
46408 JSR   46535       > Position des String feststellen, ggf. Pointer setzen
46411 BEQ   46404       Unbedingter Sprung

46413 LDA      #7       Schrittweite fuer Suche in Variablentabelle
46415 STA      83
46417 LDA      45       Pointer auf Variablentabelle
46419 LDX      46
46421 STA      34       als Suchpointer nach (34/35) bringen
46423 STX      35
46425 CPX      48       Ende der Variablentabelle erreicht?
46427 BNE   46433       Nein: weiter bei 46433
46429 CMP      47
46431 BEQ   46438       sonst zu Array-Behandlung uebergehen
46433 JSR   46525       > Position des Strings feststellen, ggf. Pointer setzen
46436 BEQ   46425       Unbedingter Sprung

46438 STA      88       Pointer in Arraytabelle
46440 STX      89
46442 LDA      #3       Schrittweite fuer Suche innnerhalb eines Arrays
46444 STA      83
46446 LDA      88
46448 LDX      89
46450 CPX      50       Ende der Arraytabelle erreicht?
46452 BNE   46461       Nein: weiter bei 46461
46454 CMP      49
46456 BNE   46461
46458 JMP   46598       > sonst Transfer

46461 STA      34       Pointer auf Arrayheader
46463 STX      35
46465 LDY      #0
46467 LDA     (34),Y    Variablenname, erstes Zeichen
46469 TAX               in XR uebertragen
46470 INY
46471 LDA     (34),Y    Variablenname, zweites Zeichen
46473 PHP               Statusflags merken
46474 INY
46475 LDA     (34),Y    Laenge des Arrays zu Pointer auf Arraytabelle addieren
46477 ADC      88
46479 STA      88
46481 INY
46482 LDA     (34),Y
46484 ADC      89
46486 STA      89
```

GARBAGE COLLECT (Fortsetzung)

```
46488 PLP
46489 BPL  46446      Stringvariable? Nein: weitersuchen ...
46491 TXA
46492 BMI  46446      Stringvariable? Nein: weitersuchen ...
46494 INY
46495 LDA    (34),Y   Anzahl Dimensionen
46497 LDY    #0
46499 ASL             mal 2 (zwei Byte je Dimension)
46500 ADC    #5       plus 5 (Arrayheader)
46502 ADC    34       zum Pointer addieren
46504 STA    34
46506 BCC  46510
46508 INC    35
46510 LDX    35       Pointer in Array
46512 CPX    89       mit Pointer auf naechstes Feld vergleichen
46514 BNE  46520      Ungleich? Ja: weiter bei 46520
46516 CMP    88
46518 BEQ  46450      Gleich? Ja: weiter mit naechstem Feld
46520 JSR  46535      > Position des Strings feststellen, ggf. Pointer setzen
46523 BEQ  46512      Unbedingter Sprung

46525 LDA    (34),Y   Variablenname, erstes Zeichen
46527 BMI  46582      Integer bzw. Function? Ja: weiter 46582
46529 INY
46530 LDA    (34),Y   Variablenname, zweites Zeichen
46532 BPL  46582      Real? Ja: weiter bei 46582
46534 INY
46535 LDA    (34),Y   Stringlaenge
46537 BEQ  46582      = 0? Ja: weiter bei 46582
46539 INY
46540 LDA    (34),Y   Startadresse des Strings nach (XR/Accu)
46542 TAX
46543 INY
46544 LDA    (34),Y
46546 CMP    52       Stringpointer mit (51/52) vergleichen
46548 BCC  46556
46550 BNE  46582      Groesser? Ja: weiter bei 46582
46552 CPX    51
46554 BCS  46582
46556 CMP    96       Stringpointer mit (95/96) vergleichen
46558 BCC  46582
46560 BNE  46566      Kleiner? Ja: weiter bei 46582
46562 CPX    95
46564 BCC  46582
46566 STX    95       Startadresse des Strings nach (95/96) bringen
46568 STA    96
46570 LDA    34       Adresse des Stringdescriptors
46572 LDX    35
46574 STA    78       nach (78/79) bringen
46576 STX    79
46578 LDA    83       Schrittweite fuer Suche in Variablentabelle
46580 STA    85
46582 LDA    83       zum Suchpointer addieren
46584 CLC
46585 ADC    34
46587 STA    34       und wieder nach (34/35) bringen
46589 BCC  46593
46591 INC    35
46593 LDX    35       (Accu/XR) := (34/35)
46595 LDY    #0       fuer unbedingten Sprung
46597 RTS
```

GARBAGE COLLECT (Fortsetzung)

```
46598 LDA     79       String zwischen Variablentabellenende
46600 ORA     78       und oberem RAM-Ende gefunden?
46602 BEQ  46593       Nein: fertig, RTS
46604 LDA     85       Falls Suchlauf in Array-Suchlauf, so ist (85) = 3,
46606 AND     #4       sonst ist (85) = 7
46608 LSR
46609 TAY              YR ist 2 bei Einzelvariable, 0 bei Array
46610 STA     85
46612 LDA   (78),Y     Laenge des Strings
46614 ADC     95       zur Anfangsadresse low des Strings addieren
46616 STA     90       ergibt Endadresse+1 low des Strings
46618 LDA     96
46620 ADC     #0
46622 STA     91       ebenso Endadresse+1 high berechnen
46624 LDA     51       Endadresse des Zielbereichs fuer Transfer
46626 LDX     52
46628 STA     88
46630 STX     89
46632 JSR  41919       > Block-Verschiebe-Routine
46635 LDY     85
46637 INY
46638 LDA     88       neue Anfangsadresse low
46640 STA   (78),Y     in Descriptor bringen
46642 TAX
46643 INC     89
46645 LDA     89       neue Anfangsadresse high
46647 INY
46648 STA   (78),Y     in Descriptor bringen
46650 JMP  46378       > weitermachen, bis alle Strings bearbeitet wurden
```

BASIC-Routine zur Stringverknuepfung (+)

```
46653 LDA    101       Pointer auf Descriptor des ersten Strings
46655 PHA              auf dem Stack zwischenspeichern
46656 LDA    100
46658 PHA
46659 JSR  44675       > Adresse des zweiten Stringdescriptors nach (100/101)
46662 JSR  44431       > FRMNUM prueft, ob Stringvariable
46665 PLA              Pointer auf Descriptor des ersten Strings
46666 STA    111       wiederherstellen
46668 PLA
46669 STA    112
46671 LDY     #0
46673 LDA  (111),Y     Laenge des ersten Strings
46675 CLC
46676 ADC  (100),Y     plus Laenge des zweiten Strings
46678 BCC  46685       Kleiner als 256? Ja: weiter bei 46685
46680 LDX    #23       Code fuer "STRING TOO LONG"
46682 JMP  42039       > Fehlermeldung, READY.

46685 JSR  46197       > Speicher pruefen, Platz fuer Gesamtstring reservieren
46688 JSR  46714       > ersten String in reservierten Bereich bringen
46691 LDA     80       Pointer auf Descriptor des zweiten Strings
46693 LDY     81       (wird bei Aufruf von 46197 nach (80/81) gebracht)
46695 JSR  46762       > FRESTR
46698 JSR  46732       > zweiten String an ersten String anhaengen
46701 LDA    111       Pointer auf Descriptor des ersten Strings
46703 LDY    112
46705 JSR  46762       > FRESTR
46708 JSR  46282       > Descriptor in Tabelle nach (25,...,33)
46711 JMP  44472       > zurueck in Auswertung von Ausdruecken
```

BASIC-Routine zur Stringverknuepfung (+)

```
46714 LDY     #0
46716 LDA   (111),Y   Stringlaenge auf Stack retten
46718 PHA
46719 INY
46720 LDA   (111),Y   Anfangsadresse low des Strings ins XR
46722 TAX
46723 INY
46724 LDA   (111),Y   Anfangsadresse high des Strings ins YR
46726 TAY
46727 PLA             Stringlaenge wieder holen
46728 STX     34      Adresse low
46730 STY     35      Adresse high
46732 TAY             Laenge = 0?
46733 BEQ  46745      Ja: weiter bei 46745
46735 PHA
46736 DEY
46737 LDA    (34),Y   String in Bereich, auf den (53/54) zeigt uebertragen
46739 STA    (53),Y
46741 TYA
46742 BNE  46736
46744 PLA             Stringlaenge
46745 CLC             Pointer (53/54) um Stringlaenge erhoehen
46746 ADC     53
46748 STA     53
46750 BCC  46754
46752 INC     54
46754 RTS
```

FRESTR: Stringverwaltungsroutine

```
46755 JSR  44431      > FRMNUM prueft, ob Stringvariable
46758 LDA    100      (100/101) zeigt auf Stringdescriptor
46760 LDY    101
46762 STA     34      nach (34/35) bringen
46764 STY     35
46766 JSR  46811      > prueft, ob identisch mit letztem String
46769 PHP             Statusflags merken
46770 LDY     #0
46772 LDA    (34),Y   Stringlaenge auf Stack retten
46774 PHA
46775 INY
46776 LDA    (34),Y   Anfangsadresse low des String ins XR
46778 TAX
46779 INY
46780 LDA    (34),Y   Anfangsadresse high des Strings ins YR
46782 TAY
46783 PLA             Stringlaenge wieder holen
46784 PLP             neuer String identisch mit altem String?
46785 BNE  46806      Nein: Anfangsadresse des Strings nach (34/35), RTS
46787 CPY     52      neue Stringadresse mit identisch mit Pointer auf
46789 BNE  46806      unteres Stringende? Nein: weiter bei 46806
46791 CPX     51
46793 BNE  46806
46795 PHA             String-Anfangspointer um Laenge des Strings
46796 CLC             hinaufsetzen
46797 ADC     51
46799 STA     51
46801 BCC  46805
46803 INC     52
```

FRESTR: Stringverwaltungsroutine (Fortsetzung)

```
46805 PLA                Stringlaenge
46806 STX      34        Startadresse low
46808 STY      35        Startadresse high
46810 RTS

46811 CPY      24        neuer Pointer auf Stringdescriptor in (Accu/YR)
46813 BNE   46827        identisch mit (23/24)? Nein: RTS
46815 CMP      23
46817 BNE   46827
46819 STA      22        neuen Pointer nach (22),
46821 SBC      #3
46823 STA      23        (23) um drei vermindern (Carry ist bereits 1)
46825 LDY      #0
46827 RTS
```

BASIC-Funktion CHR$

```
46828 JSR   47009        > GETBYT holt folgenden Wert ins XR
46831 TXA
46832 PHA                Byte merken
46833 LDA      #1        Stringlaenge auf 1 setzen
46835 JSR   46205        > Speicher pruefen, Platz reservieren
46838 PLA
46839 LDY      #0
46841 STA    (98),Y      Byte als Element des Strings speichern
46843 PLA                Ruecksprungadresse aus Stack entfernen
46844 PLA
46845 JMP   46282        > Stringdescriptor in Tabelle (25,...,33) bringen
```

BASIC-Funktion LEFT$

```
46848 JSR   46945        > Stringadresse und Parameter vom Stack holen
46851 CMP    (80),Y      Parameter fuer LEFT$ mit Stringlaenge vergleichen
46853 TYA                (Accu) := 0
46854 BCC   46860        Parameter kleiner Stringlaenge? Ja: weiter bei 46860
46856 LDA    (80),Y      Stringlaenge ins XR
46858 TAX
46859 TYA                (Accu) := 0
46860 PHA                Position des ersten Elements des neuen Strings
46861 TXA                Stringlaenge bzw. Parameter fuer LEFT$
46862 PHA
46863 JSR   46205        > Speicher pruefen, Platz reservieren
46866 LDA      80        (80/81) ist Pointer auf Stringdescriptor
46868 LDY      81
46870 JSR   46762        > FRESTR
46873 PLA
46874 TAY                Laenge des neuen Strings
46875 PLA                Position des neuen Elements (bei LEFT$ = 0)
46876 CLC                Adresse des alten Strings entsprechend erhoehen
46877 ADC      34
46879 STA      34
46881 BCC   46885
46883 INC      35
46885 TYA                Laenge des neuen Strings
46886 JSR   46732        > neuen String in Stringbereich uebertragen
46889 JMP   46282        > Stringdescriptor in Tabelle (25,...,33) bringen
```

BASIC-Funktion RIGHT$

```
46892 JSR  46945        > Stringadresse und Parameter holen
46895 CLC               zweiten Parameter von Stringlaenge subtrahieren
46896 SBC   (80),Y      (umgekehrt durch Vorzeichenumkehr)
46898 EOR   #255        Accu enthaelt Nummer des ersten Elements
46900 JMP  46854        > weiter wie bei LEFT$
```

BASIC-Funktion MID$

```
46903 LDA   #255        Ersatzwert fuer zweiten Zahlenparameter
46905 STA    101
46907 JSR    121        > CHRGOT holt letztes Zeichen
46910 CMP    #41        Code fuer ")"?
46912 BEQ  46920        Ja: kein zweiter Parameter, weiter bei 46920
46914 JSR  44797        > CHKCOM prueft, ob Komma folgt
46917 JSR  47006        > GETBYT bringt zweiten Parameter nach (101)
46920 JSR  46945        > Stringadresse und zweiten Parameter holen
46923 BEQ  47000        erster Parameter = 0? Ja: "ILLEGAL QUANTITY ERROR"
46925 DEX
46926 TXA               Position des ersten Elements innerhalb
46927 PHA               des alten Strings auf den Stack ablegen
46928 CLC
46929 LDX     #0
46931 SBC   (80),Y      Laenge des alten Strings kleiner als erster Parameter?
46933 BCS  46861        Ja: restliche Ausfuehrung bei LEFT$
46935 EOR   #255        neue Stringlaenge berechnen
46937 CMP    101        kleiner als zweiter Parameter?
46939 BCC  46862        Ja: restliche Ausfuehrung bei LEFT$
46941 LDA    101        sonst zweiten Parameter als 'rechte' Stringbegrenzung
46943 BCS  46862        Unbedingter Sprung

46945 JSR  44791        > prueft, ob ")" folgt
46948 PLA               Ruecksprungadresse low,
46949 TAY
46950 PLA               Ruecksprungadresse high der aufrufenden Routine
46951 STA     85
46953 PLA               Ruecksprungadresse low,
46954 PLA               Ruecksprungadresse high von JSR 84 (45024)
46955 PLA               erster Parameter
46956 TAX
46957 PLA               Adresse des Stringdescriptors aus Stack
46958 STA     80        nach (80/81) bringen
46960 PLA
46961 STA     81
46963 LDA     85        Ruecksprungadresse der aufrufenden Routine
46965 PHA               wieder auf den Stack legen
46966 TYA
46967 PHA
46968 LDY     #0        Index (Indirect-Adressierung) := 0
46970 TXA               erster Parameter
46971 RTS
```

BASIC-Funktion LEN

```
46972 JSR  46978        > FRESTR, String-Flag loeschen
46975 JMP  45986        > (Accu) := 0, INTFLP

46978 JSR  46755        > FRESTR
46981 LDX     #0
46983 STX     13        String-Flag loeschen
46985 TAY               Stringlaenge ins YR
46986 RTS
```

BASIC-Funktion ASC

```
46987 JSR  46978       > FRESTR, String-Flag loeschen
46990 BEQ  47000       Stringlaenge = 0? Ja: "ILLEGAL QUANTITY ERROR"
46992 LDY    #0
46994 LDA   (34),Y     erstes Zeichen des Strings
46996 TAY
46997 JMP  45986       > (Accu) := 0, INTFLP
47000 JMP  45640       > "ILLEGAL QUANTITY ERROR"
```

GETBYT: liesst Zahl im Bereich von 0 bis 255 aus BASIC Text ins XR

```
47003 JSR   115        > CHRGET holt naechstes Zeichen
47006 JSR  44426       > FRMNUM, FRMEVL numerischen Ausdruck auswerten
47009 JSR  45496       > FLPINT, falls negativ "ILLEGAL QUANTITY ERROR"
47012 LDX   100        Byte high des geholten Ausdrucks = 0?
47014 BNE  47000       Nein: Wert groesser als 255, "ILLEGAL QUANTITY ERROR"
47016 LDX   101        Byte low des geholten Ausdruck ins XR
47018 JMP   121        > CHRGOT holt letztes Zeichen
```

BASIC-Funktion VAL

```
47021 JSR  46978       > FRESTR, String-Flag loeschen
47024 BNE  47029       Stringlaenge = 0? Nein: weiter bei 47029
47026 JMP  47351       > FAC := 0, fertig

47029 LDX   122        CHRGET-Pointer
47031 LDY   123
47033 STX   113        in (113/114) aufbewahren
47035 STY   114
47037 LDX    34        String-Anfangsadresse
47039 STX   122        in CHRGET-Pointer bringen
47041 CLC
47042 ADC    34        Adresse des ersten Zeichens nach dem String
47044 STA    36        nach (36/37) bringen
47046 LDX    35
47048 STX   123
47050 BCC  47053
47052 INX
47053 STX    37
47055 LDY    #0        erstes Byte nach dem String
47057 LDA   (36),Y
47059 PHA              auf dem Stack zwischenspeichern
47060 TYA
47061 STA   (36),Y     und durch Null (als Trennungszeichen) ersetzen
47063 JSR   121        > erstes Zeichen des Strings holen
47066 JSR  48371       > String in Fliesskommazahl nach FAC bringen
47069 PLA              erstes Byte nach dem String wiederherstellen
47070 LDY    #0
47072 STA   (36),Y
47074 LDX   113        CHRGET-Pointer wiederherstellen
47076 LDY   114
47078 STX   122
47080 STY   123
47082 RTS
```

GETADR und GETBYT: Lesen einer Adresse und eines Bytes

```
47083 JSR  44426       > FRMEVL, FRMNUM wertet numerischen Ausdruck aus
47086 JSR  47095       > GETADR bringt Adresse aus FAC nach (20/21)
47089 JSR  44797       > CHKCOM prueft, ob Komma folgt
47092 JMP  47006       > GETBYT holt Byte ins XR
```

GETADR: Lesen einer Adresse im Bereich von 0 bis 65535

```
47095 LDA    102       Vorzeichen von FAC .
47097 BMI  47000       negativ? Ja: "ILLEGAL QUANTITY ERROR"
47099 LDA     97       Exponentbyte von FAC
47101 CMP   #145       groesser 16 (145 - 128 = 17)
47103 BCS  47000       Ja: "ILLEGAL QUANTITY ERROR"
47105 JSR  48283       > GK-Zahl in 16-Bit-Integerzahl ohne Vorzeichen umwandeln
47108 LDA    100       Adressbytes vertauschen
47110 LDY    101
47112 STY     20       und nach (20/21) bringen
47114 STA     21
47116 RTS
```

BASIC-Funktion PEEK

```
47117 LDA     21       (20/21) auf Stack retten, falls (20/21) von
47119 PHA              aufrufender Routine noch benoetigt wird (z. B. POKE)
47120 LDA     20
47122 PHA
47123 JSR  47095       > GETADR bringt PEEK-Adresse nach.(20/21)
47126 LDY     #0
47128 LDA    (20),Y    Inhalt der PEEK-Adresse ins YR
47130 TAY
47131 PLA              (20/21) wiederherstellen
47132 STA     20
47134 PLA
47135 STA     21
47137 JMP  45986       > (Accu) := 0, INTFLP
```

BASIC-Routine POKE

```
47140 JSR  47083       > GETADR, GETBYT
47143 TXA              Byte (zweiter Parameter)
47144 LDY     #0
47146 STA    (20),Y    in POKE-Adresse speichern
47148 RTS
```

BASIC-Routine WAIT

```
47149 JSR  47083       > GETADR, GETBYT
47152 STX     73       zweiter WAIT-Parameter nach (73)
47154 LDX     #0       Ersatzwert fuer dritten WAIT-Parameter
47156 JSR    121       > CHRGOT holt letztes Zeichen
47159 BEQ  47164       folgt Trennzeichen? Ja: weiter bei 47164
47161 JSR  47089       > CHKCOM, GETBYT
47164 STX     74       dritter WAIT-Parameter nach (74)
47166 LDY     #0
47168 LDA    (20),Y    Inhalt der WAIT-Adresse
47170 EOR     74       EXOR mit drittem Parameter
47172 AND     73       AND mit zweitem Parameter
47174 BEQ  47168       Ergebnis = 0? warten ...
47176 RTS
```

Arithmetik: FAC := 0.5 + FAC

```
47177 LDA    #17       (Accu/YR) := 48913
47179 LDY   #191
47181 JMP  47207       > FAC := FLP-Konstante + FAC
```

Arithmetik: FAC := ARG - FAC

```
47184 JSR  47756      > Konstante, auf die (Accu/YR) zeigt, nach ARG
47187 LDA    102      Vorzeichen von FAC invertieren
47189 EOR   #255
47191 STA    102
47193 EOR    110      und mit Vorzeichen von FAC verknuepfen
47195 STA    111      Ergebnis nach (111)
47197 LDA     97      Exponent von FAC (falls FAC = 0)
47199 JMP  47210      > FAC := ARG + FAC
```

Arithmetik: FAC := ARG + FAC

```
47202 JSR  47513      > Exponenten von FAC und ARG einander anpassen
47205 BCC  47267      Unbedingter Sprung
47207 JSR  47756      > Konstante, auf die (Accu/YR) zeigt, nach ARG
47210 BNE  47215      FAC = 0? Nein: weiter bei 47217
47212 JMP  48124      > FAC := ARG

47215 LDX    112      Rundungbyte fuer FAC
47217 STX     86      nach (86) bringen
47219 LDX   #105      XR als Offset-Pointer fuer ARG laden
47221 LDA    105      Exponent-Byte von ARG
47223 TAY
47224 BEQ  47176      ARG = 0? Ja: RTS
47226 SEC
47227 SBC     97      Exponentbyte von FAC subtrahieren
47229 BEQ  47267      Exponenten gleich? Ja: weiter bei 47267
47231 BCC  47251      Exponent von FAC groesser? Ja: weiter bei 47267
47233 STY     97      Exponent von FAC durch Vorzeichen von ARG ersetzen
47235 LDY    110      Vorzeichen von FAC durch Vorzeichen von ARG ersetzen
47237 STY    102
47239 EOR   #255      Vorzeichen der Exponentendifferenz wechseln
47241 ADC     #0      (Carry = 1)
47243 LDY     #0
47245 STY     86      Rundungsstelle loeschen
47247 LDX    #97      XR als Offset-Pointer fuer FAC laden
47249 BNE  47255      Unbedingter Sprung

47251 LDY     #0
47253 STY    112      Rundungsstelle von FAC loeschen
47255 CMP   #249      Differenz der Exponenten groesser 7?
47257 BMI  47202      Ja: Mantissen aneinander anpassen
47259 TAY
47260 LDA    112      Rundungsstelle von FAC
47262 LSR      1,X
47264 JSR  47536      > Angleichen durch Verschieben der Mantisse
47267 BIT    111      Vorzeichen von FAC und ARG identisch
47269 BPL  47358      Ja: Addition der Mantissen, weiter bei 47358
47271 LDY    #97      YR als Offset-Pointer fuer FAC laden
47273 CPX   #105      Ist XR als Offsetzeiger fuer ARG initialisiert?
47275 BEQ  47279      Ja: weiter bei 47279
47277 LDY   #105      YR als Offset-Pointer fuer ARG laden
47279 SEC             Mantissensubtraktion
47280 EOR   #255
47282 ADC     86      Rundungsstelle
47284 STA    112
47286 LDA      4,Y
47289 SBC      4,X
47291 STA    101
47293 LDA      3,Y
47296 SBC      3,X
47298 STA    100
```

Arithmetik: FAC := ARG + FAC (Fortsetzung)

```
47300 LDA      2,Y
47303 SBC      2,X
47305 STA     99
47307 LDA      1,Y
47310 SBC      1,X
47312 STA     98
47314 BCS  47319      Negativer Uebertrag? Nein: weiter bei 47319
47316 JSR  47431      > Mantisse invertieren

47319 LDY     #0      FAC normalisieren
47321 TYA             Accu := 0
47322 CLC
47323 LDX     98
47325 BNE  47401
47327 LDX     99
47329 STX     98
47331 LDX    100
47333 STX     99
47335 LDX    101
47337 STX    100
47339 LDX    112
47341 STX    101
47343 STY    112      Rundungsstelle loeschen
47345 ADC     #8      Zaehler fuer Bitverschiebung (8 Bits verschoben)
47347 CMP    #32      Bereits um 32 Bit verschoben?
47349 BNE  47323      Nein: zurueck zum Anfang der Schleife
47351 LDA     #0      alle Bytes der Mantisse sind gleich 0,
47353 STA     97      also ist auch FAC gleich 0. Vorzeichen und
47355 STA    102      Exponentbyte werden daher auf 0 gesetzt
47357 RTS
```

Mantissenaddition bei identischem Vorzeichen

```
47358 ADC     86      Rundungsstelle
47360 STA    112
47362 LDA    101
47364 ADC    109
47366 STA    101
47368 LDA    100
47370 ADC    108
47372 STA    100
47374 LDA     99
47376 ADC    107
47378 STA     99
47380 LDA     98
47382 ADC    106
47384 STA     98
47386 JMP  47414      > Ueberlaufbit; falls noetig, in Mantisse zurueckshiften

47389 ADC     #1      Bitzaehler erhoehen
47391 ASL    112      FAC so lange nach links verschieben, bis das
47393 ROL    101      hoechstwertigste Bit der Mantisse gesetzt ist
47395 ROL    100
47397 ROL     99
47399 ROL     98
47401 BPL  47389
47403 SEC
47404 SBC     97      Binaerexponent kleiner als Anzahl Verschiebungen?
47406 BCS  47351      Ja: Underflow, Zahl wird als 0 behandelt
47408 EOR   #255      Exponent um Anzahl der Verschiebungen vermindern
47410 ADC     #1
```

Mantissenaddition (Fortsetzung)

```
47412 STA    97
47414 BCC  47430      Ist die Carry-Flag durch Ueberlauf gesetzt? Nein: RTS
47416 INC    97       Exponent um eins erhoehen
47418 BEQ  47486      Ueberlauf im Exponenten? Ja: "OVERFLOW ERROR"
47420 ROR    98       Ueberlaufbit in Carry in Mantisse zurueckschieben,
47422 ROR    99       Carry erhaelt Position des hoechstwertigsten Bits
47424 ROR   100
47426 ROR   101
47428 ROR   112
47430 RTS
```

Vorzeichen der Mantisse invertieren (bei negativen Ergebnissen)

```
47431 LDA   102       Einserkomplement der Mantisse von FAC bilden
47433 EOR  #255
47435 STA   102
47437 LDA    98
47439 EOR  #255
47441 STA    98
47443 LDA    99
47445 EOR  #255
47447 STA    99
47449 LDA   100
47451 EOR  #255
47453 STA   100
47455 LDA   101
47457 EOR  #255
47459 STA   101
47461 LDA   112
47463 EOR  #255
47465 STA   112
47467 INC   112       Mantisse um eins erhoehen
47469 BNE  47485
47471 INC   101
47473 BNE  47485
47475 INC   100
47477 BNE  47485
47479 INC    99
47481 BNE  47485
47483 INC    98
47485 RTS

47486 LDX   #15       Code fuer "OVERFLOW"
47488 JMP  42039      > Fehlermeldung, READY.
```

Rechtsverschieben eines Registers

```
47491 LDX   #37       XR als Offset-Pointer fuer Funktionsregister laden
47493 LDY     4,X     Rechtsverschiebung um ein Byte
47495 STY   112
47497 LDY     3,X
47499 STY     4,X
47501 LDY     2,X
47503 STY     3,X
47505 LDY     1,X
47507 STY     2,X
47509 LDY   104
47511 STY     1,X
47513 ADC    #8       Bit-Zaehler um 8 erhoehen
47515 BMI  47493      Groesser 0?
47517 BEQ  47493      Nein: weiter verschieben
```

Rechtsverschieben eines Registers (Fortsetzung)

```
47519 SBC     #8      Bit-Zaehler um 8 vermindern
47521 TAY
47522 LDA    112
47524 BCS  47546      Ergebnis = 0? Ja: Fertig, CLC, RTS
47526 ASL      1,X    hoechstwertiges Bit
47528 BCC  47532      =1? Nein: weiter bei 47532
47530 INC      1,X    hoechste Mantissenstelle um eins erhoehen
47532 ROR      1,X    saemtliche Stellen um ein Bit nach recht schieben
47534 ROR      1,X
47536 ROR      2,X
47538 ROR      3,X
47540 ROR      4,X
47542 ROR
47543 INY             Bit-Zaehler um eins erhoehen
47544 BNE  47526      und weiter verschieben, bis Zaehler = 0
47546 CLC
47547 RTS
```

BASIC-Funktion LOG

Tabelle mit Fliesskommakonstanten

```
47548 129   0   0   0   0      1
47553   3                      Polynomgrad
47554 127  94  86 203 121      0.434255942   4 Koeffizienten
47559 128  19 155  11 100      0.576584541
47564 128 118  56 147  22      0.961800759
47569 130  56 170  59  32      2.88539007
47574 128  53   4 243  52      0.707106781   SQR(0.5)
47579 129  53   4 243  52      1.41421356    SQR(2)
47584 128 128   0   0   0     -0.5
47589 128  49 114  23 248      0.693147181   ln 2

47594 JSR  48171      > prueft auf Vorzeichen und 0
47597 BEQ  47601      Argument = 0? Ja: "ILLEGAL QUANTITY ERROR"
47599 BPL  47604      groesser 0? Ja: weiter bei 47604
47601 JMP  45640      > "ILLEGAL QUANTITY ERROR"
47604 LDA     97      Exponentbyte von FAC
47606 SBC   #127      128 subtrahieren (Carry ist geloescht)
47608 PHA             Exponent auf Stack retten
47609 LDA   #128
47611 STA     97      jetzt gilt: 0.5 <= FAC < 1
47613 LDA   #214      (Accu/YR) := 47574, Startadresse von SQR(0.5)
47615 LDY   #185
47617 JSR  47207      > FAC := Konstante + FAC
47620 LDA   #219      (Accu/YR) := 47579, Startadresse von SQR(2)
47622 LDY   #185
47624 JSR  47887      > FAC := Konstante / FAC
47627 LDA   #188      (Accu/YR) := 47548, Startadresse von 1
47629 LDY   #185
47631 JSR  47184      > FAC := Konstante + FAC
47634 LDA   #193      (Accu/YR) := 47553, Adresse des Polynomgrads
47636 LDY   #185
47638 JSR  57411      > Polynomauswertung
47641 LDA   #224      (Accu/YR) := 47584, Startadresse von -0.5
47643 LDY   #185
47645 JSR  47207      > FAC := Konstante + FAC
47648 PLA             Exponent vom Stack holen
47649 JSR  48510      > "Kennzahl" berechnen; Accu zu Mantisse addieren
47652 LDA   #229      (Accu/YR) := 47589, Startadresse von ln 2
47654 LDY   #185
```

Arithmetik: FAC := ARG * FAC

```
47656 JSR   47756       > Konstante, auf die (Accu/YR) zeigt, nach ARG
47659 BNE   47664       FAC = 0? Nein: weiter bei 47664
47661 JMP   47755       > FAC := ARG

47664 JSR   47799       > Exponent des Ergebnisses durch Addition berechnen
47667 LDA      #0       Register fuer Funktionen initialisieren
47669 STA      38
47671 STA      39
47673 STA      40
47675 STA      41
47677 LDA     112
47679 JSR   47705       > Multiplikation
47682 LDA     101
47684 JSR   47705       > Multiplikation
47687 LDA     100
47689 JSR   47705       > Multiplikation
47692 LDA      99
47694 JSR   47705       > Multiplikation
47697 LDA      98
47699 JSR   47710       > Multiplikation
47702 JMP   48015       > Register fuer Funktionen nach FAC, FAC normalisieren

47705 BNE   47710
47707 JMP   47491       > Register fuer Funktionen nach rechts verschieben

47710 LSR
47711 ORA    #128
47713 TAY
47714 BCC   47741       Binaere Multiplikation des Accus mit ARG, Ergebnis ins
47716 CLC               Register fuer Funktionen. Fuer jedes gesetzte Bit im
47717 LDA      41       Accu wird ARG zum Funktionsregister addiert, unabhaengig
47719 ADC     109       davon wird das Funktionsregister verdoppelt.
47721 STA      41       (Prinzip siehe auch 45900)
47723 LDA      40
47725 ADC     108
47727 STA      40
47729 LDA      39
47731 ADC     107
47733 STA      39
47735 LDA      38
47737 ADC     106
47739 STA      38
47741 ROR      38
47743 ROR      39
47745 ROR      40
47747 ROR      41
47749 ROR     112
47751 TYA
47752 LSR
47753 BNE   47713
47755 RTS
```

Konstante, auf die (Accu/YR) zeigt, nach ARG, Arithmetik vorbereiten

```
47756 STA      34       Konstante, auf die (Accu/YR) zeigt, nach ARG
47758 STY      35
47760 LDY      #4
47762 LDA     (34),Y
47764 STA     109
47766 DEY
47767 LDA     (34),Y
```

Arithmetik: FAC := ARG * FAC (Fortsetzung)

```
47769 STA    108
47771 DEY
47772 LDA    (34),Y
47774 STA    107
47776 DEY
47777 LDA    (34),Y
47779 STA    110       Vorzeichen von FAC und ARG verknuepfen
47781 EOR    102
47783 STA    111
47785 LDA    110
47787 ORA   #128
47789 STA    106
47791 DEY
47792 LDA    (34),Y
47794 STA    105
47796 LDA     97       Exponent von FAC (als Kennzeichnung, falls FAC = 0)
47798 RTS

47799 LDA    105       Exponent von ARG = 0?
47801 BEQ  47834       Ja: Ruecksprungadresse vom Stack holen, FAC := 0, RTS
47803 CLC
47804 ADC     97       Exponenten von FAC und ARG addieren
47806 BCC  47812
47808 BMI  47839       Ueberlauf im Exponenten? Ja: "OVERFLOW ERROR"
47810 CLC
47811 BIT    ...
47812 BPL  47836       Underflow? Ja: FAC := 0, RTS
47814 ADC   #128
47816 STA     97       ergibt Exponenten von FAC
47818 BNE  47823
47820 JMP  47355       > FAC := 0, RTS

47823 LDA    111       Verknuepfung der Vorzeichen von FAC und ARG
47825 STA    102       als Vorzeichen des Ergebnisses in FAC speichern
47827 RTS

47828 LDA    102       Vorzeichen positiv? (vgl. 57355)
47830 EOR   #255
47832 BMI  47839       Ja: "OVERFLOW ERROR"
47834 PLA              Underflow: Ruecksprungadresse vom Stack
47835 PLA
47836 JMP  47351       > FAC := 0, RTS

47839 JMP  47486       > "OVERFLOW ERROR"
```

Arithmetik: FAC := 10 * FAC

```
47842 JSR  48140       > FAC runden, ARG := FAC
47845 TAX              Exponent von FAC
47846 BEQ  47864       FAC = 0? Ja: RTS
47848 CLC
47849 ADC     #2       Binaerexponent um zwei erhoehen (entspricht 4 * FAC)
47851 BCS  47839       Ueberlauf? Ja: OVERFLOW ERROR
47853 LDX     #0
47855 STX    111
47857 JSR  47223       > FAC := ARG + FAC (mit obigem Ergebnis also * 5)
47860 INC     97       Binaerexponent um eins erhoehen (entspricht 2 * FAC)
47862 BEQ  47839       Ueberlauf? Ja: "OVERFLOW ERROR"
47864 RTS

47865 132  32   0   0   0      Konstante 10
```

Arithmetik: FAC := FAC / 10

```
47870 JSR  48140       > FAC runden, ARG := FAC
47873 LDA   #249       (Accu/YR) := 47865, Startadresse von 10
47875 LDY   #186
47877 LDX     #0
47879 STX    111
47881 JSR  48034       > FAC mit Konstante, auf die (Accu/YR) zeigt, laden
47884 JMP  47890       > FAC := ARG / FAC
```

Arithmetik: FAC := ARG / FAC

```
47887 JSR  47756       > Konstante, auf die (Accu/YR) zeigt, nach ARG
47890 BEQ  48010       FAC = 0? Ja: "DIVISION BY ZERO ERROR"
47892 JSR  48155       > FAC runden
47895 LDA     #0
47897 SEC
47898 SBC     97       Vorzeichen des Exponenten von FAC wechseln
47900 STA     97
47902 JSR  47799       > Exponent und Vorzeichen des Ergebnisses bestimmen
47905 INC     97
47907 BEQ  47839       Ueberlauf im Exponenten? Ja: "OVERFLOW ERROR"
47909 LDX   #252       Pointer auf Funktionsregister (in Zero-Page zyklisch!)
47911 LDA     #1
47913 LDY    106       FAC mit ARG byteweise vergleichen
47915 CPY     98
47917 BNE  47935
47919 LDY    107
47921 CPY     99
47923 BNE  47935
47925 LDY    108
47927 CPY    100
47929 BNE  47935
47931 LDY    109
47933 CPY    101
47935 PHP              Statusregister auf Stack retten
47936 ROL
47937 BCC  47948
47939 INX
47940 STA    41,X      Ergebnis in (38,...,42) aufbauen
47942 BEQ  47994       XR gleich 0? Ja: weiter bei 47994
47944 BPL  47998       XR gleich 1? Ja: fertig, weiter bei 47998
47946 LDA     #1
47948 PLP
47949 BCS  47965       FAC <= ARG? Ja: weiter bei 47965
47951 ASL    109       ARG verdoppeln
47953 ROL    108
47955 ROL    107
47957 ROL    106
47959 BCS  47935       Ueberlauf? Ja: weiter 47935
47961 BMI  47913       hoechstwertiges Bit gesetzt? Ja: weiter bei 47913
47963 BPL  47935       sonst weiter bei 47935
47965 TAY              Mantisse von ARG minus Mantisse von FAC
47966 LDA    109
47968 SBC    101
47970 STA    109
47972 LDA    108
47974 SBC    100
47976 STA    108
47978 LDA    107
47980 SBC     99
47982 STA    107
47984 LDA    106
```

Arithmetik: FAC := ARG / FAC (Fortsetzung)

```
47986 SBC     98
47988 STA    106
47990 TYA
47991 JMP  47951

47994 LDA    #64
47996 BNE  47948        Unbedingter Sprung

47998 ASL               (Accu) := 64 * (Accu)
47999 ASL
48000 ASL
48001 ASL
48002 ASL
48003 ASL
48004 STA    112        ergibt Rundungsstelle
48006 PLP
48007 JMP  48015        > Ergebnis nach FAC, FAC normalisieren
48010 LDX    #20        Code fuer "DIVISION BY ZERO"
48012 JMP  42039        > Fehlermeldung, READY.

48015 LDA     38        Register fuer Funktionen nach FAC uebertragen
48017 STA     98
48019 LDA     39
48021 STA     99
48023 LDA     40
48025 STA    100
48027 LDA     41
48029 STA    101
48031 JMP  47319        > FAC linksbuendig machen
```

Konstante, auf die (Accu/YR) zeigt, nach FAC (Speicherformat in Registerformat)

```
48034 STA     34        (Accu/YR) nach (34/35)
48036 STY     35
48038 LDY     #4
48040 LDA    (34),Y     LSB der Mantisse
48042 STA    101
48044 DEY
48045 LDA    (34),Y
48047 STA    100
48049 DEY
48050 LDA    (34),Y
48052 STA     99
48054 DEY
48055 LDA    (34),Y     MSB der Mantisse (Bit 7 ist Vorzeichenflag)
48057 STA    102        Register fuer Vorzeichen von FAC
48059 ORA   #128        Bit 7 (hoechstwertiges Bit) auf eins setzen
48061 STA     98        in Register fuer MSB der Mantisse von FAC bringen
48063 DEY
48064 LDA    (34),Y     Exponentbyte
48066 STA     97
48068 STY    112        Rundungsstelle loeschen
48070 RTS
```

Uebertragung von FAC an andere Stelle

```
48071 LDX    #92        Zielbereich (92,...,96)
48073 BIT    ...
48074 LDX    #87        Zielbereich (87,...,91)
48076 LDY     #0        Pointer auf Zielbereich high
48078 BEQ  48084        Unbedingter Sprung
```

Uebertragung von FAC an andere Stelle (Fortsetzung)

```
48080 LDX     73       Startadresse des Zielbereichs in (73/74)
48082 LDY     74

48084 JSR  48155       > FAC runden
48087 STX     34       Zeiger auf Zielbereich setzen
48089 STY     35
48091 LDY     #4       FAC in Zielbereich uebertragen
48093 LDA    101       LSB der Mantisse
48095 STA    (34),Y
48097 DEY
48098 LDA    100
48100 STA    (34),Y
48102 DEY
48103 LDA     99
48105 STA    (34),Y
48107 DEY
48108 LDA    102       Vorzeichenbyte fuer FAC
48110 ORA   #127       Bits 0 bis 6 fuer "AND" auf eins setzen
48112 AND     98       und MSB der Mantisse hineinbringen
48114 STA    (34),Y    (MSBit wird durch Vorzeichenflag ersetzt)
48116 DEY
48117 LDA     97       Exponentbyte von FAC
48119 STA    (34),Y
48121 STY    112       Rundungstelle von FAC loeschen
48123 RTS
```

Uebertragung von ARG nach FAC

```
48124 LDA    110       Vorzeichenbyte von ARG
48126 STA    102       in Register fuer Vorzeichen von FAC uebertragen
48128 LDX     #5
48130 LDA    104,X
48132 STA     96,X
48134 DEX
48135 BNE  48130
48137 STX    112       Rundungstelle von FAC loeschen
48139 RTS
```

Uebertragung von FAC nach ARG

```
48140 JSR  48155       > FAC runden
48143 LDX     #6
48145 LDA     96,X
48147 STA    104,X
48149 DEX
48150 BNE  48145
48152 STX    112       Rundungsstelle von FAC loeschen
48154 RTS
```

FAC runden

```
48155 LDA     97       Exponent = 0 (also FAC = 0)?
48157 BEQ  48154       Ja: RTS
48159 ASL    112       Rundungsstelle < 128?
48161 BCC  48154       Ja: RTS
48163 JSR  47471       > Mantisse um eins erhoehen
48166 BNE  48154       Matisse jetzt null? Nein: RTS
48168 JMP  47416       > Mantisse nach rechts schieben, Exponent erhoehen
```

Vorzeichen von FAC pruefen

```
48171 LDA      97       Ergebnis der Pruefung:
48173 BEQ   48184
48175 LDA     102       | FAC  |Accu  |C-Bit|Z-Bit|N-Bit|
48177 ROL               ------------------------------
48178 LDA    #255       | > 0  |    1 |  0  |  0  |  0  |
48180 BCS   48184       | = 0  |    0 | unv |  1  |  0  |
48182 LDA      #1       | < 0  | 255  |  1  |  0  |  1  |
48184 RTS
```

BASIC-Funktion SGN

```
48185 JSR   48171       > Vorzeichen von FAC pruefen (1, 0, 255 fuer +, 0, -)
48188 STA      98
48190 LDA      #0
48192 STA      99
48194 LDX    #136       Exponentbyte fuer Ergebnis
48196 LDA      98
48198 EOR    #255
48200 ROL
48201 LDA      #0
48203 STA     101
48205 STA     100
48207 STX      97       Exponentbyte fuer FAC
48209 STA     112
48211 STA     102
48213 JMP   47314       > Invertieren, falls negativ und linksbuendig machen
```

BASIC-Funktion ABS

```
48216 LSR     102       > Bit 7 im Vorzeichenbyte von FAC loeschen
48218 RTS
```

Zahlenvergleich zwischen FAC und Konstante, auf die (Accu/YR) zeigt

```
48219 STA      36       Pointer auf Konstante setzen
48221 STY      37
48223 LDY      #0
48225 LDA    (36),Y     Exponentbyte der Konstanten
48227 INY
48228 TAX
48229 BEQ   48171       Exponent = 0? Ja: Vorzeichen von FAC pruefen
48231 LDA    (36),Y     hoechste Stelle der Konstanten
48233 EOR     102       Vorzeichenbyte von FAC
48235 BMI   48175       Vorzeichen unterschiedlich? Ja: weiter bei 48175
48237 CPX      97       Exponenten vergleichen
48239 BNE   48274       unterschiedlich? Ja: weiter bei 48274
48241 LDA    (36),Y
48243 ORA    #128       Vorzeichenbit in Vergleichzahl setzen
48245 CMP      98       MSBs vergleichen
48247 BNE   48274
48249 INY
48250 LDA    (36),Y
48252 CMP      99       zweite Stelle vergleichen
48254 BNE   48274
48256 INY
48257 LDA    (36),Y
48259 CMP     100       dritte Stelle vergleichen
48261 BNE   48274
48263 INY
48264 LDA    #127
48266 CMP     112       Rundungsstelle von FAC mit 127 vergleichen
```

Zahlenvergleich zwischen FAC und Konstante (Fortsetzung)

```
48268 LDA    (36),Y   Letzte Stellen voneinander subtrahieren
48270 SBC     101     mit Carry gemaess Vergleich der Rundungsstelle
48272 BEQ   48314     Alle Stellen gleich? Ja: RTS
48274 LDA     102     Vorzeichen von FAC
48276 BCC   48280     Konstante < FAC? Ja: weiter bei 48280
48278 EOR    #255     Accu invertieren
48280 JMP   48177     > Vergleichsroutine fuer 0
```

Umwandlung einer Fliesskommazahl in Integer

```
48283 LDA      97     Exponent von FAC
48285 BEQ   48361     = 0? Ja: weiter bei 48361
48287 SEC
48288 SBC    #160     Integer-Exponent
48290 BIT     102     FAC positiv?
48292 BPL   48303     Ja: weiter bei 48303
48294 TAX
48295 LDA    #255
48297 STA     104
48299 JSR   47437     > Mantisse von FAC invertieren
48302 TXA
48303 LDX     #97     XR als Offset-Pointer auf FAC
48305 CMP    #249     Exponent groesser als -8?
48307 BPL   48315     Ja: weiter bei 48315
48309 JSR   47513     > FAC nach rechts verschieben, bis Exponent = 0
48312 STY     104
48314 RTS

48315 TAY
48316 LDA     102
48318 AND    #128     Vorzeichen isolieren
48320 LSR      98
48322 ORA      98
48324 STA      98
48326 JSR   47536     > FAC nach rechts verschieben (mit Carry)
48329 STY     104
48331 RTS
```

BASIC-Funktion INT

```
48332 LDA      97     Exponentbyte von FAC
48334 CMP    #160     groesser oder gleich 160 (FAC also sowieso ganzzahlig)?
48336 BCS   48370     Ja: RTS
48338 JSR   48283     > Fliesskommazahl in Integer umwandeln
48341 STY     112     Rundungsstelle loeschen
48343 LDA     102     Vorzeichen in Accu bringen
48345 STY     102     Vorzeichen positiv machen
48347 EOR    #128
48349 ROL             Carry bei negativem Vorzeichen loeschen
48350 LDA    #160
48352 STA      97
48354 LDA     101
48356 STA       7
48358 JMP   47314     > FAC linksbuendig machen

48361 STA      98     Mantisse von FAC mit Nullen fuellen
48363 STA      99     (siehe 48285)
48365 STA     100
48367 STA     101
48369 TAY
48370 RTS
```

STRFAC, Umwandlung eines Zahlenstrings in eine Fliesskommazahl

Eingabe: CHRGET-Pointer auf erstes Zeichen des Zahlenstring,
         erstes Zeichen im Accu

```
48371 LDY     #0        (93,...,103) mit 0 vorbesetzen
48373 LDX    #10
48375 STY     93,X
48377 DEX
48378 BPL  48375
48380 BCC  48397        erstes Zeichen Ziffer? Ja: weiter bei 48397
48382 CMP    #45        Code fuer "-"?
48384 BNE  48390        Nein: weiter bei 48390
48386 STX    103        (103) := 255, Vorzeichenflag setzen
48388 BEQ  48394        Unbedingter Sprung
48390 CMP    #43        Code fuer "+"
48392 BNE  48399        Nein: weiter bei 48399
48394 JSR    115        > CHRGET holt naechstes Zeichen
48397 BCC  48490        Ziffer? Ja: weiter bei 48490
48399 CMP    #46        Code fuer "."?
48401 BEQ  48449        Ja: weiter bei 48449
48403 CMP    #69        Code fuer "E"?
48405 BNE  48455        Nein: weiter bei 48455
48407 JSR    115        > CHRGET holt naechstes Zeichen
48410 BCC  48435        Ziffer? Ja: weiter 48435, weiter bei 48529
48412 CMP   #171        BASIC-Code fuer "-"?
48414 BEQ  48430        Ja: weiter bei 48430
48416 CMP    #45        ASCII fuer "-"?
48418 BEQ  48430        Ja: weiter bei 48430
48420 CMP   #170        BASIC-Code fuer "+"?
48422 BEQ  48432        Ja: weiter bei 48432
48424 CMP    #43        ASCII fuer "+"?
48426 BEQ  48432        Ja: weiter bei 48432
48428 BNE  48437

48430 ROR     96        Bit 7 von (96) setzen
48432 JSR    115        > CHRGET holt naechstes Zeichen
48435 BCC  48529        Ziffer? Ja: weiter bei 48529
48437 BIT     96        Bit 7 von (96) gesetzt?
48439 BPL  48455        Nein: weiter bei 48455
48441 LDA     #0
48443 SEC
48444 SBC     94        Negativer Exponent: Vorzeichen des Exponenten wechseln
48446 JMP  48457

48449 ROR     95        Aufruf durch Dezimalpunkt
48451 BIT     95        bereits zweiter Dezimalpunkt?
48453 BVC  48394        Nein: weitermachen ...
48455 LDA     94        gelesene Zahl gemaess Position
48457 SEC
48458 SBC     93        des Dezimalpunkts
48460 STA     94        und Exponenten anpassen
48462 BEQ  48482
48464 BPL  48475
48466 JSR  47870        > FAC := FAC / 10
48469 INC     94
48471 BNE  48466
48473 BEQ  48482
```

Umwandlung eines Zahlenstrings in eine Fliesskommazahl (Fortsetzung)

```
48475 JSR   47842       > FAC := FAC * 10
48478 DEC      94       gelesene Zahl gemaess gelesenem Exponenten anpassen
48480 BNE   48475
48482 LDA     103       Gelesene Zahl negativ?
48484 BMI   48487       Ja: Vorzeichen invertieren
48486 RTS

48487 JMP   49076       > Vorzeichen von FAC wechseln

48490 PHA               Aufruf durch Mantissenziffer
48491 BIT      95       Vorkommastelle?
48493 BPL   48497       Ja: weiter bei 48497
48495 INC      93       Zaehler fuer Anzahl Nachkommastellen erhoehen
48497 JSR   47842       > FAC := FAC * 10
48500 PLA
48501 SEC
48502 SBC     #48       ASCII in Ziffernwert von 0 bis 9 umwandeln
48504 JSR   48510       > Ziffer zur Mantisse von FAC addieren
48507 JMP   48394       > naechstes Zeichen holen

48510 PHA
48511 JSR   48140       > ARG := FAC
48514 PLA
48515 JSR   48188       > Accu in hoechste Stelle von FAC bringen
48518 LDA     110       Vorzeichen von FAC
48520 EOR     102       und Vorzeichen von ARG
48522 STA     111       miteinander verknuepfen
48524 LDX      97
48526 JMP   47210       > FAC := ARG + FAC

48529 LDA      94       Aufruf durch Exponentenziffer
48531 CMP     #10       dritte Exponentziffer?
48533 BCC   48544       Nein: weiter bei 48544
48535 LDA    #100
48537 BIT      96       Vorzeichen negativ?
48539 BMI   48558       Ja: Undeflow, gelesene Zahl als Null behandeln
48541 JMP   47486       > "OVERFLOW ERROR"

48544 ASL               Exponent mit 10 multiplizieren
48545 ASL
48546 CLC
48547 ADC      94
48549 ASL
48550 CLC
48551 LDY      #0
48553 ADC   (122),Y     und neue Exponentziffer addieren
48555 SEC
48556 SBC     #48
48558 STA      94
48560 JMP   48432       > naechste Ziffer holen
```

Tabelle mit Fliesskommakonstanten

```
48563 155  62 188  31 253        99999999.9
48568 158 110 107  39 253       999999999
48573 158 110 107  40   0      1000000000
```

Ausgabe der Zeilennummer bei Fehlermeldungen, 'LIST' etc. (Einsprung 48589)

```
48578 LDA   #113      (Accu/YR) := 41841, Startadresse von " IN "
48580 LDY   #163
48582 JSR  48602      > Ausgabe von " IN "
48585 LDA     58      laufende Zeilennummer fuer ERROR
48587 LDX     57
48589 STA     98      in die beiden hoechsten Stellen von FAC
48591 STX     99
48593 LDX   #144      Exponent := 16 (144 - 128 = 16)
48595 SEC
48596 JSR  48201      > uebrige Stellen von FAC mit Null fuellen
48599 JSR  48607      > FACSTR wandelt FAC in String ab (256) um
48602 JMP  43806      > String drucken
```

Umwandlung von FAC in einen Zahlenstring

```
48605 LDY     #1      Zeiger auf Stringbereich
48607 LDA    #32      Code fuer Space (Vorzeichen fuer positive Zahl)
48609 BIT    102      Vorzeichen von FAC
48611 BPL  48615      positiv? Ja: weiter bei 48615
48613 LDA    #45      Code fuer "-"
48615 STA    255,Y    in erste Stelle des Stringbereichs bringen
48618 STA    102
48620 STY    113
48622 INY
48623 LDA    #48      ASCII fuer Null
48625 LDX     97      Exponent = 0?
48627 BNE  48632      Nein: weiter bei 48632
48629 JMP  48900      > ASCII-Null nach 257, Endezeichen nach 258, RTS

48632 LDA     #0
48634 CPX   #128      Exponentbyte von FAC = 128?
48636 BEQ  48640      Ja: 0.5 <= FAC < 1, weiter bei 48640
48638 BCS  48649      FAC >= 1? Ja: weiter bei 48649
48640 LDA   #189      (Accu/YR) := 48573, Startadresse von 1E+09
48642 LDY   #189
48644 JSR  47656      > FAC := Konstante * FAC
48647 LDA   #247      entspricht -9
48649 STA     93      (93) = -9, falls FAC < 1, sonst (93) = 0
48651 LDA   #184      (Accu/YR) := 48568, Startadresse von 999999999
48653 LDY   #189
48655 JSR  48219      > FAC mit Konstante vergleichen
48658 BEQ  48690
48660 BPL  48680
48662 LDA   #179      (Accu/YR) := 48563, Startadresse von 99999999.9
48664 LDY   #189
48666 JSR  48219      > FAC mit Konstante vergleichen
48669 BEQ  48673
48671 BPL  48687
48673 JSR  47842      > FAC := 10 * FAC
48676 DEC     93      Dezimalexponent vermindern
48678 BNE  48662
48680 JSR  47870      > FAC := FAC / 10
48683 INC     93      Dezimalexponent erhoehen
48685 BNE  48651
48687 JSR  47177      > FAC := 0.5 + FAC
48690 JSR  48283      > Fliesskommazahl nach Integer umwandeln
48693 LDX     #1      FAC liegt nun im Bereich von 1E8 bis 1E9
48695 LDA     93      (93) enthaelt den Betrag der
48697 CLC             korrigierenden Zehnerpotenz
48698 ADC    #10      Betrag der Zahl < 0.01?
48700 BMI  48711      Ja: weiter bei 48711
```

Umwandlung von FAC in einen Zahlenstring (Fortsetzung)

```
48702 CMP    #11       Betrag der Zahl > 1E9?
48704 BCS  48712       Ja: weiter bei 48712
48706 ADC   #255
48708 TAX
48709 LDA     #2
48711 SEC
48712 SBC     #2
48714 STA     94       Exponentialdarstellungsflag (=0 wenn .01 <= Betrag < 1E9)
48716 STX     93       Negativdarstellung des Exponenten
48718 TXA
48719 BEQ  48723       0.1 <= Betrag < 1? Ja: weiter bei 48723
48721 BPL  48742       0.01 <= Betrag < 0.1? Nein: weiter bei 48742
48723 LDY    113
48725 LDA    #46       ASCII fuer "."
48727 INY
48728 STA  255,Y       in Stringbereich bringen
48731 TXA
48732 BEQ  48740       0.1 <= Betrag < 1? Ja: weiter bei 48740
48734 LDA    #48       ASCII fuer "0"
48736 INY
48737 STA  255,Y       in Stringbereich bringen
48740 STY    113
48742 LDY     #0       Pointer in Tabelle mit Stellenwerten
48744 LDX   #128
48746 LDA    101       Durch wechselnde Addition und Subtraktion
48748 CLC              der Stellenwerte aus der Konstantentabelle
48749 ADC 48921,Y      werden die einzelnen Ziffern des Zahlenstrings
48752 STA    101       berechnet.
48754 LDA    100
48756 ADC 48920,Y
48759 STA    100
48761 LDA     99
48763 ADC 48919,Y
48766 STA     99
48768 LDA     98
48770 ADC 48918,Y
48773 STA     98
48775 INX
48776 BCS  48782
48778 BPL  48746
48780 BMI  48784
48782 BMI  48746
48784 TXA
48785 BCC  48791       Komplement addiert? Nein: weiter bei 48791
48787 EOR   #255       Ergebnis bezueglich 10 komplementieren
48789 ADC    #10
48791 ADC    #47       ergibt Ziffercode
48793 INY              Tabellenzeiger auf naechste Konstante fuer
48794 INY              Stellenwerte setzen
48795 INY
48796 INY
48797 STY     71
48799 LDY    113       Zeiger auf Stringbereich
48801 INY              auf naechste Zifferposition erhoehen
48802 TAX
48803 AND   #127       Zifferncode
48805 STA  255,Y       in Stringbereich bringen
48808 DEC     93
48810 BNE  48818       Einerstelle erreicht? Nein: weiter bei 48818
```

Umwandlung von FAC in einen Zahlenstring (Fortsetzung)

```
48812 LDA    #46       ASCII fuer "."
48814 INY
48815 STA    255,Y     in Stringbereich bringen
48818 STY    113       Pointer merken
48820 LDY     71       Tabellenzeiger fuer Stellenwerte wiederherstellen
48822 TXA
48823 EOR   #255
48825 AND   #128
48827 TAX
48828 CPY    #36       Ende der Fliesskomma-Stringumwandlung?
48830 BEQ  48836       Ja: weiter bei 48836
48832 CPY    #60       Ende der Umwandlung von 60stel Sekunden in TI$?
48834 BNE  48746       Nein: bei naechster Stelle weitermachen
48836 LDY    113
48838 LDA    255,Y     letzte von Null verschiedene Stelle suchen
48841 DEY
48842 CMP    #48       Code fuer "0"?
48844 BEQ  48838       Ja: weiter bei 48838
48846 CMP    #46       Code fuer "."?
48848 BEQ  48851       Ja: weiter bei 48851
48850 INY
48851 LDA    #43       Code fuer "+" (Exponentialdarstellung)
48853 LDX     94       Flag fuer Exponentialdarstellung gesetzt?
48855 BEQ  48903       Nein: weiter bei 48903
48857 BPL  48867       Zehnerexponent positiv? Ja: weiter bei 48867
48859 LDA     #0       Betrag des Exponenten berechnen
48861 SEC
48862 SBC     94
48864 TAX
48865 LDA    #45       Code fuer "-"
48867 STA    257,Y     in Stringbereich bringen
48870 LDA    #69       Code fuer "E"
48872 STA    256,Y     in Stringbereich bringen
48875 TXA              ASCII fuer Zehnerstelle des Exponenten berechnen
48876 LDX    #47
48878 SEC
48879 INX
48880 SBC    #10
48882 BCS  48879
48884 ADC    #58       Code fuer Einerstelle des Exponenten berechnen
48886 STA    259,Y     in Stringbereich bringen
48889 TXA
48890 STA    258,Y     Code fuer Zehnerstelle in Stringbereich bringen
48893 LDA     #0       Endezeichen
48895 STA    260,Y     hinter letztes Zeichen setzen
48898 BEQ  48908       Unbedingter Sprung

48900 STA    255,Y
48903 LDA     #0       Endezeichen
48905 STA    256,Y     hinter letztes Zeichen setzen
48908 LDA     #0       (Accu/YR) := 256, Startadresse fuer String
48910 LDY     #1
48912 RTS

48913 128   0   0   0   0      Gleitkommakonstante 0.5
```

Tabelle mit Stellenwerten bei bei Umwandlung von Fliesskomma in String

```
48918 250  10  31   0     -100000000   Dezimale Stellenwerte im Integerformat
48922   0 152 150 128      100000000   zur Berechnung der Ziffern in der
48926 255 240 189 192       -1000000   Dezimaldarstellung
48930   0   1 134 160         100000
48934 255 255 216 240         -10000
48938   0   0   3 232           1000
48942 255 255 255 156           -100
48946   0   0   0  10             10
48950 255 255 255 255             -1

48954 255 223  10 128       -2160000   Entsprechende Stellenwerte zur
48958   0   3  75 192         216000   Berechnung der Ziffern fuer TI$
48962 255 255 115  96         -36000
48966   0   0  14  16           3600
48970 255 255 253 168           -600
48974   0   0   0  60             60

48978 236      ?

48979 170 170 170 170 170 170 170 170 170 170
      ...
      ...
48999 170 170 170 170 170 170 170 170 170 170
```

BASIC-Funktion SQR

```
49009 JSR  48140       > FAC runden und nach ARG uebertragen
49012 LDA    #17       (Accu/YR) := 48913, Startadresse von 0.5
49014 LDY   #191
```

Arithmetik: FAC := ARG ↑ FAC

```
49016 JSR  48034       > Konstante, auf die (Accu/YR) zeigt, nach FAC
49019 BEQ  49133       FAC = 0? Ja: weiter bei 49133
49021 LDA    105       Exponentbyte von ARG
49023 BNE  49028       = 0? Nein: weiter bei 49028
49025 JMP  47353       > FAC := 0

49028 LDX    #78
49030 LDY     #0
49032 JSR  48084       > FAC nach (78,...,82) bringen
49035 LDA    110       Vorzeichen von ARG
49037 BPL  49054       positiv? Ja: weiter bei 49054
49039 JSR  48332       > INT schneidet Nachkommastellen von FAC ab
49042 LDA    #78
49044 LDY     #0
49046 JSR  48219       > FAC mit (78,...,82) vergleichen
49049 BNE  49054       Gleich (Exponent ganzzahlig)? Nein: weiter bei 49054
49051 TYA              (Accu) := 4 (Wert aus Subroutine 48219)
49052 LDY      7       letzte Exponentstelle
49054 JSR  48126       > Betrag von FAC nach ARG
49057 TYA
49058 PHA              letzte Exponentstelle
49059 JSR  47594       > LOG, FAC := ln FAC
49062 LDA    #78
49064 LDY     #0
49066 JSR  47656       > FAC := (78,...,82) * FAC
49069 JSR  49133       > FAC := exp FAC
49072 PLA              letzte Exponentstelle
49073 LSR              Exponent gradzahlig?
49074 BCC  49086       Ja: RTS
```

Arithmetik: FAC := ARG ↑ FAC (Fortsetzung)

```
49076 LDA      97       FAC = 0?
49078 BEQ   49086       Ja: RTS
49080 LDA     102       FAC := - FAC
49082 EOR    #255
49084 STA     102
49086 RTS
```

BASIC-Funktion EXP

Konstantentabelle fuer EXP

```
49087 129   56 170  59  41       1.44269504      1 / ln 2
49092   7                        Polynomgrad
49093 113   52  88  62  86       2.14987637E-05 Koeffizienten
49098 116   22 126 179  27       1.43523140E-04
49103 119   47 238 227 133       1.34226348E-03
49108 122   29 132  28  42       9.61401701E-03
49113 124   99  89  88  10        .0555051269
49118 126 117 253 231 198        .240226385
49123 128   49 114  24  16        .693147186     ln 2
49128 129    0   0   0   0       1
```

```
49133 LDA    #191       (Accu/YR) := 49087, Startadresse von 1.44269504
49135 LDY    #191
49137 JSR   47656       > FAC := Konstante * FAC
49140 LDA     112       Rundungsstelle
49142 ADC     #80       plus 80
49144 BCC   49149       kleiner 256? Ja: weiter bei 49149
49146 JSR   48163       > Mantisse von FAC um eins erhoehen
49149 JMP   57344       > Fortsetzung in zweitem ROM-Bereich
```

BASIC-Funktion EXP (Fortsetzung aus erstem ROM-Bereich)

```
57344 STA     86      Rundungsstelle + 80
57346 JSR  48143      > ARG := FAC
57349 LDA     97      Exponentbyte von FAC
57351 CMP   #136
57353 BCC  57358      FAC < 128? Ja: weiter bei 57358
57355 JSR  47828      > wenn positiv "OVERFLOW ERROR", sonst Underflow, FAC:=0

57358 JSR  48332      > INT schneidet Nachkommastellen ab
57361 LDA      7      Ganzzahliger Anteil
57363 CLC
57364 ADC   #129
57366 BEQ  57355      FAC = 127? Ja: weiter bei 57355
57368 SEC
57369 SBC     #1
57371 PHA
57372 LDX     #5      FAC mit ARG vertauschen
57374 LDA   105,X
57376 LDY    97,X
57378 STA    97,X
57380 STY   105,X
57382 DEX
57383 BPL  57374
57385 LDA     86
57387 STA    112      Rundungsstelle
57389 JSR  47187      > FAC := ARG - FAC
57392 JSR  49076      > FAC := -FAC
57395 LDA   #196      (Accu/YR) := 49092, Startadresse fuer Polynomauswertung
57397 LDY   #191
57399 JSR  57433      > Polynomauswertung
57402 LDA     #0
57404 STA    111
57406 PLA
57407 JSR  47801      > Exponenten von ARG und FAC addieren
57410 RTS
```

Polynomauswertung

```
57411 STA    113      Polynomberechnung a0*x+a1*x↑3+a2*x↑5+a3*x↑7+...
57413 STY    114      Adresse des Polynomgrads zwischenspeichern
57415 JSR  48074      > FAC runden und nach (87,...,91) uebertragen
57418 LDA    #87
57420 JSR  47656      > FAC := (87,...,91) * FAC
57423 JSR  57437      > Polynomauswertung
57426 LDA    #87      (Accu/YR) := 87
57428 LDY     #0
57430 JMP  47656      > FAC := (87,...,91) * FAC

57433 STA    113      Polynomberechnung a0+a1*x+a2*x↑2+a3*x↑3+a4*x↑4...
57435 STY    114      Adresse des Polynomgrads zwischenspeichern
57437 JSR  48071      > FAC nach (92,...,96) uebertragen
57440 LDA  (113),Y
57442 STA    103      Polynomgrad als Zaehler fuer Polynomauswertung
57444 LDY    113      Zeiger auf Anfang der Koeffiziententabelle setzen
57446 INY
57447 TYA
57448 BNE  57452
57450 INC    114
57452 STA    113
57454 LDY    114
57456 JSR  47656      > FAC := Koeffizient (Pointer in (Accu/YR)) * FAC
```

Polynomauswertung (Fortsetzung)

```
57459 LDA    113
57461 LDY    114
57463 CLC               Pointer auf Koeffiziententabelle auf
57464 ADC     #5        naechstes Element setzen
57466 BCC  57469
57468 INY
57469 STA    113
57471 STY    114
57473 JSR  47207        > FAC := Koeffizient + FAC
57476 LDA    #92
57478 LDY     #0
57480 DEC    103        Zaehler fuer Polynomauswertung vermindern
57482 BNE  57456        alle Koeffizienten verrechnet? Nein: weitermachen ...
57484 RTS
```

BASIC-Funktion RND

```
57485 152  53  68 122   0      11879546
57490 104  40 177  70   0      3.92767774E-08

57495 JSR  48171        > Vorzeichen von FAC pruefen
57498 BMI  57555        negativ? Ja: weiter bei 57555
57500 BNE  57534        Gleich Null? Nein: weiter bei 57534
57502 JSR  65523        > IOBASE holt Startadresse der IRQ-CIA
57505 STX     34
57507 STY     35
57509 LDY     #4        Offset-Pointer setzen
57511 LDA    (34),Y     Werte aus Timer A
57513 STA     98
57515 INY
57516 LDA    (34),Y
57518 STA    100
57520 LDY     #8        sowie die Zehntel- und ganzen Sekunden
57522 LDA    (34),Y     aus "Time Of Day" nach FAC uebertragen
57524 STA     99
57526 INY
57527 LDA    (34),Y
57529 STA    101
57531 JMP  57571        > ueberspringen bis 57571

57534 LDA   #139        (Accu/YR) := 139, Startadresse der letzten Zufallszahl
57536 LDY     #0
57538 JSR  48034        > (139,....,143) nach FAC uebertragen
57541 LDA   #141        (Accu/YR) := 57485, Startadresse von Konstante
57543 LDY   #224
57545 JSR  47656        > FAC := Konstante * FAC
57548 LDA   #146        (Accu/YR) := 57490, Startadresse von Konstante
57550 LDY   #224
57552 JSR  47207        > FAC := Konstante + FAC
57555 LDX    101        Stellen in FAC vertauschen
57557 LDA     98
57559 STA    101
57561 STX     98
57563 LDX     99
57565 LDA    100
57567 STA     99
57569 STX    100
57571 LDA     #0
57573 STA    102        Vorzeichen positiv machen
57575 LDA     97        Exponenten
57577 STA    112        in Rundungsstelle bringen
```

BASIC-Funktion RND (Fortsetzung)

```
57579 LDA   #128        Zufallszahl in Bereich von [ 0;1 [ bringen
57581 STA     97
57583 JSR  47319        > FAC linksbuendig machen
57586 LDX   #139        (Accu/YR) := 139, Startadresse der letzten Zufallszahl
57588 LDY     #0
57590 JMP  48084        > FAC runden und nach (139,...,143) bringen
```

Ansprungadressen und Errorhandling der KERNAL-Routinen und I/O-Befehle

```
57593 CMP   #240        Accu = 240?
57595 BNE  57604        Nein: weiter bei 57604
57597 STY     56        (XR/YR) als Pointer auf Ende Arbeitsspeicher
57599 STX     55        (Pufferbereich fuer RS-232 Uebertragung schuetzen)
57601 JMP  42595        > CLR ohne CLALL
57604 TAX               Fehlercode im Accu = 0?
57605 BNE  57609        Nein: Fehlermeldung ausgeben, weiter bei 57609
57607 LDX    #30        Code fuer "BREAK"
57609 JMP  42039        > Fehlermeldung, READY.

57612 JSR  65490        > CHROUT, Zeichen auf aktiven Kanal ausgeben
57615 BCS  57593        Fehler? Ja: weiter bei 57593
57617 RTS

57618 JSR  65487        > CHRIN, Zeichen vom aktiven Kanal holen
57621 BCS  57593        Fehler? Ja: weiter bei 57593
57623 RTS

57624 JSR  58541        > CHKOUT, Ausgabevorbereitungen
57627 BCS  57593        Fehler? Ja: weiter bei 57593
57629 RTS

57630 JSR  65478        > CHKIN, Vorbereitungen fuer Datenempfang
57633 BCS  57593        Fehler? Ja: weiter bei 57593
57635 RTS

57636 JSR  65508        > GETIN, Zeichen von Tastatur in Accu
57639 BCS  57593        Fehler? Ja: weiter bei 57593
57641 RTS
```

BASIC-Routine SYS

```
57642 JSR  44426        > FRMEVL, FRMNUM
57645 JSR  47095        > GETADR bringt SYS-Argument nach (20/21)
57648 LDA   #225        Ruecksprungadresse (57671 - 1) auf Stack legen
57650 PHA
57651 LDA    #70
57653 PHA
57654 LDA    783        Adresse (783) entspricht dem Statusregister
57657 PHA
57658 LDA    780        Datenuebergabe-Parameter fuer SYS von
57661 LDX    781        (780,...,782) in die Register Accu, XR, YR bringen
57664 LDY    782
57667 PLP               Wert aus Adresse (783) ins Statusregister bringen
57668 JMP   (20)        Aufruf der SYS-Routine
57671 PHP               Statusregister retten
57672 STA    780        Accu, XR, YR als Ergebnisse der SYS-Routine
57675 STX    781        wieder nach (780,...,782) speichern
57678 STY    782
57681 PLA               Statusflags nach (783) bringen
57682 STA    783
57685 RTS
```

BASIC-Routine SAVE

```
57686 JSR  57812      > Parameter fuer SAVE lesen
57689 LDX    45       Endadresse+1 fuer SAVE
57691 LDY    46
57693 LDA   #43       Offset fuer 16-Bit-Pointer auf Startadresse in (43/44)
57695 JSR  65496      > SAVE, speichern von Programmen
57698 BCS  57593      Fehler? Ja: weiter bei 57593
57700 RTS
```

BASIC-Routine LOAD/VERIFY

```
57701 LDA    #1       Einsprung fuer VERIFY
57703 BIT   ...
57704 LDA    #0       Einsprung fuer LOAD
57706 STA    10       entsprechendes Flag setzen
57708 JSR  57812      > Parameter fuer LOAD/VERIFY lesen
57711 LDA    10       Flag fuer 'VERIFY'
57713 LDX    43       Zeiger auf Anfang des BASIC-Bereichs
57715 LDY    44       fuer Append nach (XR/YR) bringen
57717 JSR  65493      > LOAD, Load und Verify von Programmen
57720 BCS  57809      Fehler? Ja: weiter bei 57809
57722 LDA    10       Verify-Flag
57724 BEQ  57749      gesetzt? Nein: weiter bei 57749
57726 LDX   #28       Code fuer "VERIFY"
57728 JSR  65463      > READST, I/O-Status in Accu bringen
57731 AND   #16       Bit 4 (Nichtuebereinstimmung) gesetzt?
57733 BNE  57758      Ja: Ausgabe "VERIFY ERROR", weiter bei 57758
57735 LDA   122       CHRGET-Pointer low (sollte wohl 123 (high) sein)
57737 CMP    #2       = 2? (Direktmodusabfrage, bei low jedoch unsinnig)
57739 BEQ  57748      Ja: RTS (sollte BNE sein)
57741 LDA  #100       (Accu/YR) := 41828, Startadresse von "OK"
57743 LDY  #163
57745 JMP  43806      > Ausgabe der Meldung

57748 RTS

57749 JSR  65463      > READST, I/O-Status in Accu bringen
57752 AND  #191       Bit fuer End or Identify loeschen
57754 BEQ  57761      Sonstige Bits gesetzt? Nein: weiter bei 57761
57756 LDX   #29       Code fuer "LOAD"
57758 JMP  42039      > Ausgabe Fehlermeldung, READY.

57761 LDA   123       CHRGET-Pointer high
57763 CMP    #2       = 2 (Direktmodus)?
57765 BNE  57781      Nein: weiter bei 57781
57767 STX    45       Zeiger auf Ende des Programms setzen
57769 STY    46
57771 LDA  #118       (Accu/YR) := 41846, Startadresse von "READY."
57773 LDY  #163
57775 JSR  43806      > Meldung ausgeben
57778 JMP  42282      > CLR, Linkpointer setzen, Eingabewarteschleife

57781 JSR  42638      > CHRGET-Pointer ruecksetzen
57784 JSR  42291      > Linkpointer setzen
57787 JMP  42615      > RESTORE: Reset Descriptorindex, Stack; CONT sperren
```

BASIC-Routine OPEN

```
57790 JSR  57881      > Parameter fuer OPEN lesen
57793 JSR  65472      > OPEN, spezifiziertes File oeffnen
57796 BCS  57809      Fehler aufgetreten? Ja: weiter bei 57593
57798 RTS
```

BASIC-Routine CLOSE

```
57799 JSR  57881       > Parameter fuer CLOSE lesen
57802 LDA     73       Filenummer in Accu bringen
57804 JSR  65475       > CLOSE, File (Nummer im Accu) schliessen
57807 BCC  57748       Fehler aufgetreten? Nein: RTS
57809 JMP  57593       > I/O-Fehlerbehandlung
```

Parameter fuer 'LOAD', 'SAVE' und 'VERIFY' lesen

```
57812 LDA     #0       Laenge Filename (Ersatzwert)
57814 JSR  65469       > SETNAM, Festsetzung Parameter fuer Filename
57817 LDX     #1       Ersatzparameter fuer Primaeradresse (Recorder)
57819 LDY     #0       Ersatzparameter fuer Sekundaeradresse
57821 JSR  65466       > SETLFS, Festsetzung Parameter fuer OPEN
57824 JSR  57862       > zurueck zur uebergeordneten Routine, falls Trennzeichen
57827 JSR  57942       > Filenamen lesen
57830 JSR  57862       > zurueck zur uebergeordneten Routine, falls Trennzeichen
57833 JSR  57856       > naechsten Parameter (Geraetenummer) lesen
57836 LDY     #0       Ersatzparameter fuer Sekundaeradresse
57838 STX     73
57840 JSR  65466       > SETLFS, Festsetzung Parameter fuer OPEN
57843 JSR  57862       > zurueck zur uebergeordneten Routine, falls Trennzeichen
57846 JSR  57856       > naechsten Parameter (Sekundaeradresse) lesen
57849 TXA
57850 TAY
57851 LDX     73       Geraetenummer
57853 JMP  65466       > SETLFS, Festsetzung Parameter fuer OPEN

57856 JSR  57870       > prueft auf Komma und Nichttrennzeichen
57859 JMP  47006       > GETBYT liest Zahl ins XR

57862 JSR    121       > CHRGOT holt letztes Zeichen
57865 BNE  57869       Trennzeichen? Nein: RTS
57867 PLA              sonst Ruecksprung zur uebergeordneten Routine
57868 PLA
57869 RTS

57870 JSR  44797       > CHKCOM prueft, ob Komma folgt
57873 JSR    121       > CHRGOT holt letztes Zeichen
57876 BNE  57869       Trennzeichen? Nein: RTS
57878 JMP  44808       > "SYNTAX ERROR"
```

Parameter fuer 'OPEN' und 'CLOSE' lesen

```
57881 LDA     #0       Laenge Filename als Ersatzparameter
57883 JSR  65469       > SETNAM, Festsetzung Parameter fuer Filenamen
57886 JSR  57873       > prueft auf Komma und Nicht-Endezeichen
57889 JSR  47006       > GETBYT liest Filenummer ins XR
57892 STX     73
57894 TXA              Filenummer in Accu bringen
57895 LDX     #1       Ersatzparameter fuer Geraetenummer (Recorder)
57897 LDY     #0       Ersatzparameter fuer Sekundaeradresse (lesen)
57899 JSR  65466       > SETLFS, Festsetzung Parameter fuer OPEN
57902 JSR  57862       > zurueck zur uebergeordneten Routine, falls Trennzeichen
57905 JSR  57856       > naechsten Parameter (Geraetenummer) lesen
57908 STX     74
57910 LDY     #0       Ersatzparameter fuer Sekundaeradresse
57912 LDA     73       eingegebene Filenummer
57914 CPX     #3       Geraetenummer kleiner 3?
57916 BCC  57919       Ja: weiter bei 57919
57918 DEY              Ersatzparameter fuer Sekundaeradresse := 255
57919 JSR  65466       > SETLFS, Festsetzung Parameter fuer OPEN
```

Parameter fuer 'OPEN' und 'CLOSE' lesen (Fortsetzung)

```
57922 JSR   57862       > zurueck zur uebergeordneten Routine, falls Trennzeichen
57925 JSR   57856       > naechsten Parameter (Sekundaeradresse) lesen
57928 TXA
57929 TAY               Sekundaeradresse ins YR
57930 LDX      74       Geraetenummer in XR
57932 LDA      73       Filenummer in Accu
57934 JSR   65466       > SETLFS, Festsetzung Parameter fuer OPEN
57937 JSR   57862       > zurueck zur uebergeordneten Routine, falls Trennzeichen
57940 JSR   57870       > prueft auf Komma und Nicht-Endezeichen
57943 JSR   44446       > FRMEVL, Auswertung von Ausdruecken
57946 JSR   46755       > FRESTR, prueft, ob Stringvariable; Stringverwaltung
57949 LDX      34       Startadresse des Filenamens (Laenge im Accu)
57951 LDY      35
57953 JMP   65469       > SETNAM, Festsetzung Parameter fuer Filenamen
```

BASIC-Funktion COS

```
57956 LDA    #224       (Accu/YR) := 58080, Startadresse von PI/2
57958 LDY    #226
57960 JSR   47207       > FAC := Konstante + FAC
```

BASIC-Funktion SIN

```
57963 JSR   48140       > FAC nach ARG uebertragen
57966 LDA    #229       (Accu/YR) := 58085, Startadresse von 2*PI, Periodenlaenge
57968 LDY    #226
57970 LDX     110       Vorzeichen von ARG als Wert fuer Vorzeichenverknuepfung
57972 JSR   47879       > FAC := FAC / Konstante
57975 JSR   48140       > FAC nach ARG uebertragen
57978 JSR   48332       > INT, Nachkommastellen abschneiden
57981 LDA      #0
57983 STA     111       Wert fuer Vorzeichenverknuepfung
57985 JSR   47187       > FAC := ARG - FAC (ergibt Nachkommastellen, Argument
                          wurde in Wert innerhalb des Hauptdefinitionsbereichs
                          transformiert, FAC enthaelt Wert geteilt durch 2*PI)
57988 LDA    #234       (Accu/YR) := 58090, Startadresse von 0.25
57990 LDY    #226
57992 JSR   47184       > FAC := Konstante - FAC
57995 LDA     102       Vorzeichen von FAC
57997 PHA
57998 BPL   58013       positiv? Ja: weiter bei 58013
58000 JSR   47177       > FAC := 0.5 + FAC
58003 LDA     102       Vorzeichen von FAC
58005 BMI   58016       negativ? Ja: weiter bei 58016
58007 LDA      18       Flag fuer Vorzeichen invertieren
58009 EOR    #255
58011 STA      18
58013 JSR   49076       > FAC := -FAC
58016 LDA    #234       (Accu/YR) := 58090, Startadresse von 0.25
58018 LDY    #226
58020 JSR   47207       > FAC := Konstante + FAC
58023 PLA
58024 BPL   58029
58026 JSR   49076       > FAC := -FAC
58029 LDA    #239       (Accu/YR) := 58095, Startadresse fuer Polynomauswertung
58031 LDY    #226
58033 JMP   57411       > Polynomauswertung
```

BASIC-Funktion TAN

```
58036 JSR  48074       > FAC nach (87,...,91) bringen
58039 LDA     #0
58041 STA     18       Vorzeichenflag auf positiv setzen
58043 JSR  57963       > SIN
58046 LDX    #78       (XR/YR) := 78
58048 LDY     #0
58050 JSR  57590       > FAC runden und nach (78,...,82) bringen
58053 LDA    #87       (XR/YR) := 87
58055 LDY     #0
58057 JSR  48034       > (87,...,91) nach FAC bringen
58060 LDA     #0
58062 STA    102       Vorzeichen von FAC
58064 LDA     18       Vorzeichenflag in Accu
58066 JSR  58076       > FAC := cos (FAC)
58069 LDA    #78
58071 LDY     #0
58073 JMP  47887       > FAC := (78,...,82) / FAC

58076 PHA
58077 JMP  58013
```

Konstanten fuer trigonometrische Funktionen

```
58080 129  73  15 218 162       1.57079633   PI/2
58085 131  73  15 218 162       6.28318531   PI*2
58090 127   0   0   0   0        .25
58095   5                                    Polynomgrad
58096 132 230  26  45  27     -14.3813907
58101 134  40   7 251 248      42.0077971
58106 135 153 104 137   1     -76.7041703
58111 135  35  53 223 225      81.6052237
58116 134 165  93 231  40     -41.3417021
58121 131  73  15 218 162       6.28318531   PI*2
```

BASIC-Funktion von ATN

```
58126 LDA    102       Vorzeichen von FAC
58128 PHA
58129 BPL  58134       positiv? Nein: weiter bei 58134
58131 JSR  49076       > FAC := -FAC
58134 LDA     97       Exponentbyte von FAC
58136 PHA
58137 CMP   #129       FAC kleiner 1?
58139 BCC  58148       Ja: weiter bei 58148
58141 LDA   #188       (Accu/YR) := 47548, Startadresse von 1
58143 LDY   #185
58145 JSR  47887       > FAC := 1 / FAC
58148 LDA    #62       (Accu/YR) := 58174, Startadresse fuer Polynomauswertung
58150 LDY   #227
58152 JSR  57411       > Polynomauswertung
58155 PLA
58156 CMP   #129       Argument kleiner 1?
58158 BCC  58167       Ja: weiter bei 58167
58160 LDA   #224       (Accu/YR) := 58080, Startadresse von PI/2
58162 LDY   #226
58164 JSR  47184       > FAC := Konstante - FAC
58167 PLA              Vorzeichen des Arguments
58168 BPL  58173       positiv? Ja: RTS
58170 JMP  49076       > FAC := -FAC
58173 RTS
```

Konstanten fuer ATN

```
58174 11                                        Polynomgrad
58175 118 179 131 189 211      -6.84793912E-04   Koeffizienten
58180 121  30 244 166 245       4.85094216E-03
58185 123 131 252 176  16      -0.0161117018
58190 124  12  31 103 202       0.034209638
58195 124 222  83 203 193      -0.0542791328
58200 125  20 100 112  76       0.0724571965
58205 125 183 234  81 122      -0.0898023954
58210 125  99  48 136 126       0.110932413
58215 126 146  68 153  58      -0.142839808
58220 126  76 204 145 199       0.19999912
58225 127 170 170 170  19      -0.333333316
58230 129   0   0   0   0       1

58235 JSR  65484       > CLRCHN, aktive I/O-Kanaele schliessen
58238 LDA     #0
58240 STA     19       Tastatur als aktiven Eingabe-Kanal
58242 JSR  42618       > Descriptorindex und Stack ruecksetzen; CONT sperren
58245 CLI
58246 LDX   #128       Code fuer Ausgabe von "READY."
58248 JMP   (768)      Normalwert des Vektors (768/769): 58251
58251 TXA              Fehlercode in Accu (Statusflags setzen)
58252 BMI  58257       Bit 7 gesetzt? Ja: weiter bei 58257
58254 JMP  42042       > Ausgabe Fehlermeldung, Eingabewarteschleife

58257 JMP  42100       > Ausgabe "READY.", Eingabewarteschleife
```

BASIC-Reset

```
58260 JSR  58451       > Sprungvektortabelle (768,...,779) initialisieren
58263 JSR  58303       > Vektoren etc. fuer BASIC initialisieren
58266 JSR  58402       > Ausgabe Reset-Meldung
58269 LDX   #251
58271 TXS              Stackpointer initialisieren
58272 BNE  58246       Unbedingter Sprung zur Ausgabe von "READY.", etc.
```

CHRGET-Routine (wird zusammen mit RND-Wert nach (115,...,143) kopiert)

Zero-Flag ist gesetzt, falls Trennzeichen (0 oder 58) gelesen wurde
Carry ist geloescht, falls Ziffer (48 bis 57) gelesen wurde

```
58274 INC    122       CHRGET-Pointer low erhoehen
58276 BNE  58280
58278 INC    123       CHRGET-Pointer high erhoehen
58280 LDA  60000       hier steht die zu bearbeitende Adresse fuer BASIC
58283 CMP    #58       gelesener Wert > 57 (keine Zahl)?
58285 BCS  58297       Ja: RTS
58287 CMP    #32       Leercode?
58289 BEQ  58274       Ja: naechstes Zeichen lesen
58291 SEC
58292 SBC    #48       setzt Carry, falls Code < 48
58294 SEC
58295 SBC   #208
58297 RTS
```

Zufallszahl fuer RND nach dem Einschalten

```
58298 128  79 199  82  88      .811635157
```

Sprungvektoren, Pointer und anderes fuer BASIC initialisieren

```
58303 LDA    #76       Code fuer JMP in
58305 STA     84       Vektor fuer Auswertung von Funktionen,
58307 STA    784       Vektor fuer BASIC-Funktion USR bringen
58310 LDA    #72       (Accu/YR) := 45640, Ausgabe von "ILLEGAL QUANTITY ERROR"
58312 LDY   #178
58314 STA    785       als Kennzeichnung fuer nichtdefinierte USR-Funktion
58317 STY    786
58320 LDA   #145       (Accu/YR) := 45969, Startadresse fuer INTFLP
58322 LDY   #179
58324 STA      5       in Vektor fuer INTFLP bringen
58326 STY      6
58328 LDA   #170       (Accu/YR) := 45482, Startadresse fuer FLPINT,
58330 LDY   #177
58332 STA      3       in Vektor fuer FLPINT bringen
58334 STY      4
58336 LDX    #28
58338 LDA  58274,X     CHRGET und Zufallszahl fuer RND von (58274,...,58292)
58341 STA    115,X     nach (115,...,143) bringen
58343 DEX
58344 BPL  58338
58346 LDA     #3
58348 STA     83       Schrittweite fuer GARBAGE COLLECT
58350 LDA     #0
58352 STA    104
58354 STA     19       Tastatur als aktiven Eingabe-Kanal
58356 STA     24       Byte high des Pointers auf letzten String
58358 LDX     #1
58360 STX    509       Linkpointer zum Einbau von Programmzeilen
58363 STX    508       in BASIC-Bereich (vgl. 42274)
58366 LDX    #25       Descriptorindex initialisieren
58368 STX     22
58370 SEC
58371 JSR  65436       > MEMBOT, Lesen des Speicheranfangs
58374 STX     43       und nach (43/44) bringen
58376 STY     44
58378 SEC
58379 JSR  65433       > MEMTOP, Lesen des Speicherendes
58382 STX     55       nach (55/56) bringen
58384 STY     56
58386 STX     51       und als Stringpointer nach (51/52) bringen
58388 STY     52
58390 LDY     #0       Code Null
58392 TYA
58393 STA    (43),Y    in erste Stelle des Arbeitsspeichers bringen
58395 INC     43       und Zeiger auf Anfang BASIC um eins erhoehen
58397 BNE  58401
58399 INC     44
58401 RTS
```

Ausgabe Reset-Meldung

```
58402 LDA     43       Pointer auf Anfang BASIC-Bereich
58404 LDY     44
58406 JSR  41992       > Test, ob genug Platz im Arbeitsspeicher
58409 LDA   #115       (Accu/YR) := 58483, Startadresse von "COMMODORE 64 ..."
58411 LDY   #228
58413 JSR  43806       > Meldung ausgeben
58416 LDA     55       Pointer auf Ende des Arbeitsspeichers
58418 SEC
58419 SBC     43       minus Pointer auf Beginn des Arbeitsspeichers
58421 TAX
```

Ausgabe Reset-Meldung (Fortsetzung)

```
58422 LDA      56
58424 SBC      44       ergibt Anzahl freie Bytes
58426 JSR   48589       > Zahl ausgeben
58429 LDA     #96       (Accu/YR) := 58464, Startadresse von "BASIC BYTES ..."
58431 LDY    #228
58433 JSR   43806       > Meldung ausgeben
58436 JMP   42564       > NEW
```

Normalwerte der Vektoren von 768 bis 779

```
58439 139 227   58251   Vektor fuer ERROR
58441 131 164   42115   Vektor fuer BASIC-Eingabeschleife nach "READY."
58443 124 165   42364   Vektor fuer Umwandlung in Tokens
58445  26 167   42778   Vektor fuer Ausgabe von Tokens (LIST)
58447 228 167   42980   Vektor fuer Interpreterschleife
58449 134 174   44678   Auswertung von Ausdruecken (FRMEVL)

58451 LDX     #11
58453 LDA  58439,X      Vektoren von (768,...,779) initialisieren
58456 STA    768,X
58459 DEX
58460 BPL  58453
58462 RTS

58463   0
58464  32  66  65  83  73  67  32  66  89  84  69  83  32       BASIC BYTES
58477  70  82  69  69  13   0                                   FREE
58483 147  13  32  32  32  32  42  42  42  42  32               ****
58494  67  79  77  77  79  68  79  82  69  32  54  52  32       COMMODORE 64
58507  66  65  83  73  67  32  86  50  32  42  42  42  42  13   BASIC V2 ****
58521  13  32  54  52  75  32  82  65  77  32                   64K RAM
58531  83  89  83  84  69  77  32  32   0                       SYSTEM
58540  92                                                       ?

58541 PHA               Accu merken
58542 JSR   65481       > CHKOUT, Ausgabevorbereitungen
58545 TAX               eventuellen Fehlercode ins XR
58546 PLA               Accu wiederherstellen
58547 BCC   58550       Fehler? Nein: RTS
58549 TXA               Fehlercode wiederherstellen
58550 RTS

58551 170 170 170 170 170 170 170 170 170 170
      ...
      ...
58576 170 170 170 170 170 170 170 170 170 170

58586 LDA   53281       Background-Color als Farbwert
58589 STA   (243),Y     der Cursorposition speichern
58591 RTS

58592 ADC      #2       (Accu + Carry + 2) * 256 / 60 Sekunden warten
58594 LDY     145       Flag fuer diverse Tasten in YR bringen
58596 INY               Taste gedrueckt?
58597 BNE   58603       Ja: RTS
58599 CMP     161       Zeit vorbei?
58601 BNE   58594       Nein: warten ...
58603 RTS

58604  25  38  68  25  26  17 232  13 112  12   Timerkonstantentabelle fuer PAL
58614   6   6 209   2  55   1 174   0 105   0   vgl. 62508 ff, 65218 ff
```

IOBASE, Uebergabe der Startadresse der IRQ-CIA in (XR/YR)

```
58624 LDX    #0       (XR/YR) := 56320
58626 LDY  #220
58628 RTS
```

SCREEN, Uebergabe des Bildschirmformats in (XR/YR)

```
58629 LDX   #40
58631 LDY   #25
58633 RTS
```

PLOT, Lesen und Setzen der Cursorposition

```
58634 BCS  58643       Carry gesetzt? Ja: weiter bei 58643
58636 STX    214       Cursorzeile,
58638 STY    211       Cursorspalte setzen
58640 JSR  58732       > zugehoerige Zeilenparameter aktualisieren
58643 LDX    214       Cursorzeile,
58645 LDY    211       Cursorspalte lesen
58647 RTS
```

Screen-Editor-Reset

```
58648 JSR  58784       > aktive I/O-Kanaele und Video-Chip initialisieren
58651 LDA     #0
58653 STA    657       CBM-Taste (Kleinschrift/Graphik-Umschaltung) freigeben
58656 STA    207       Cursor in Dunkelphase
58658 LDA    #72
58660 STA    655       (655/656) := 60232, Vektor fuer Tastaturabfrage
58663 LDA   #235
58665 STA    656
58668 LDA    #10
58670 STA    649       Maximale Groesse des Tastaturpuffers
58673 STA    652       Zaehler fuer Anspruchszeit von REPEAT
58676 LDA    #14
58678 STA    646       Momentaner Cursorfarbwert
58681 LDA     #4
58683 STA    651       Zaehler fuer Wiederholungszeit von REPEAT
58686 LDA    #12
58688 STA    205       Zaehler fuer Cursorblinkdauer
58690 STA    204       Cursor ausschalten

58692 LDA    648       CLEAR SCREEN, Bildschirm loeschen
58695 ORA   #128       Startadresse high des Bildschirms
58697 TAY              Bit 7 setzen (Einfachzeilenkennzeichnung)
58698 LDA     #0
58700 TAX
58701 STY    217,X     in Tabelle fuer MSB (Doppelzeilentabelle) bringen
58703 CLC
58704 ADC    #40       Zeilenlaenge addieren
58706 BCC  58709       Uebertrag? Nein: MSB nicht erhoehen
58708 INY
58709 INX
58710 CPX    #26       alle 25 Zeilen initialisiert?
58712 BNE  58701       Nein: weitermachen ...
58714 LDA   #255       sollte wohl Kennzeichnung der '26. Zeile' als Einfach-
58716 STA    217,X     zeile sein, ist jedoch bereits vorher geschehen.
58718 LDX    #24       Bildschirm loeschen
58720 JSR  59903       Zeile (Nummer im XR) loeschen
58723 DEX
58724 BPL  58720
```

HOME, Cursor in linke obere Ecke

```
58726 LDY     #0
58728 STY    211       Cursorspalte := 0
58730 STY    214       Cursorzeile := 0
```

Zeilenparameter aktualisieren

```
58732 LDX    214       Cursorzeile ins XR
58734 LDA    211       Cursorspalte in Accu
58736 LDY    217,X     MSBs fuer Doppelzeilen
58738 BMI  58748       Einfachzeile? Ja: weiter bei 58748
58740 CLC
58741 ADC    #40       Cursorspalte um 40 erhoehen
58743 STA    211
58745 DEX              um eine Zeile zurueckgehen
58746 BPL  58736       und weiter zugehoerige Startzeile der Doppelzeile suchen
58748 LDA    217,X     (209/210) auf erstes Zeichen der Cursorzeile setzen
58750 AND     #3
58752 ORA    648
58755 STA    210       high
58757 LDA  60656,X     Tabelle der LSBs fuer Bildschirm
58760 STA    209       low
58762 LDA    #39       Zeilenlaenge fuer Einfachzeile
58764 INX
58765 LDY    217,X     Einfachzeile?
58767 BMI  58775       Ja: Zeilenlaenge speichern, RTS
58769 CLC
58770 ADC    #40       40 addieren
58772 INX              eine Zeile vorgehen
58773 BPL  58765       und weiter nach letzter Zeile der Doppelzeile suchen
58775 STA    213       Zeilenlaenge speichern
58777 RTS

58778 JSR  58784       > aktive I/O-Kanaele und Video-Chip initialisieren
58781 JMP  58726       > HOME

58784 LDA     #3       aktiven Ausgabe-Kanal
58786 STA    154       auf Bildschirm setzen
58788 LDA     #0       aktiven Eingabe-Kanal
58790 STA    153       auf Tastatur setzen
58792 LDX    #47
58794 LDA  60600,X
58797 STA  53247,X     Register des Video-Controllers setzen
58800 DEX
58801 BNE  58794
58803 RTS
```

Tastaturpuffer abbauen

```
58804 LDY    631       erstes Zeichen des Keyboardbuffers
58807 LDX     #0
58809 LDA    632,X     Zeichen in Keyboardbuffer
58812 STA    631,X     um eine Stelle nach unten verschieben
58815 INX
58816 CPX    198       bis alle gueltigen Zeichen erreicht
58818 BNE  58809
58820 DEC    198       Zaehler fuer Anzahl Zeichen um eins vermindern
58822 TYA              Accu enthaelt nun Zeichen aus Tastaturpuffers
58823 CLI
58824 CLC
58825 RTS
```

Warteschleife fuer CHRIN

```
58826 JSR  59158      > Ausgabe eines Zeichens auf den Bildschirm
58829 LDA    198      Anzahl Zeichen im Tastaturpuffer
58831 STA    204      Cursor ein/ausschalten
58833 STA    658      Insert Enable (bei Zeilenuebergang neue Zeile einfuegen)
58836 BEQ  58829      keine Zeichen im Tastaturpuffer? Ja: weiter warten
58838 SEI             Interrupt verhindern
58839 LDA    207      Cursor hell?
58841 BEQ  58855      Nein: weiter bei 58855
58843 LDA    206      Zeichen unter Cursor
58845 LDX    647      Farbe unter Cursor
58848 LDY     #0
58850 STY    207      Cursor dunkel schalten
58852 JSR  59923      > Zeichen in Bildschirm-RAM bringen
58855 JSR  58804      > Zeichen aus Keyboardbuffer holen
58858 CMP   #131      Code fuer "Shift-RUNSTOP"?
58860 BNE  58878      Nein: weiter bei 58878
58862 LDX     #9
58864 SEI
58865 STX    198      9 Zeichen in Tastaturpuffer bringen
58867 LDA  60646,X    "LOAD" CR "RUN" CR
58870 STA    630,X
58873 DEX
58874 BNE  58867
58876 BEQ  58829

58878 CMP    #13      RETURN?
58880 BNE  58826      Nein: zurueck zur Eingabeschleife
```

Zeichenuebernahme vom Bildschirm

```
58882 LDY    213      Laenge der momentanen Zeile
58884 STY    208      Flag fuer "RETURN" setzen
58886 LDA  (209),Y    Vom Ende her Zeichen vom Bildschirm lesen
58888 CMP    #32      gleich Space?
58890 BNE  58895      Nein: weiter bei 58895
58892 DEY             Zaehler vermindern
58893 BNE  58886      und weitersuchen
58895 INY             Zaehler erhoehen und als
58896 STY    200      Zeiger hinter letztes Zeichen der Zeile speichern
58898 LDY     #0      Insert Disable
58900 STY    658      (bei Zeilenuebergang kein Einfuegen von Zeilen)
58903 STY    211      Cursorposition := 0
58905 STY    212      Flag fuer Anfuehrungszeichen ruecksetzen
58907 LDA    201      Cursorzeile beim Aufruf von CHRIN bereits durch
58909 BMI  58938      Scrolling verschwunden? Ja: weiter bei 58938
58911 LDX    214      Cursorzeile ins XR
58913 JSR  59117      > Adresse fuer zugehoerige Startzeile nach (209/210)
58916 CPX    201      wurde waehrend der Eingabe die Anfangszeile verlassen?
58918 BNE  58938      Ja: ab erster Spalte der neuen Zeile lesen
58920 LDA    202      Cursorspalte bei Aufruf von CHRIN
58922 STA    211      in Cursorspaltenpointer bringen
58924 CMP    200      mit Index fuer Anzahl gueltiger Zeichen vergleichen
58926 BCC  58938      Kleiner: Zeile auswerten, weiter bei 58938
58928 BCS  58973      Groesser oder gleich: Leere Eingabe, weiter bei 58973
```

Einstieg fuer CHRIN vom Bildschirm

```
58930 TYA              Indexregister retten
58931 PHA
58932 TXA
58933 PHA
58934 LDA    208       noch Zeichen vorhanden (Zeile nicht komplett gelesen)?
58936 BEQ   58829      Nein:: zurueck zur Warteschleife, weiter bei 58829
58938 LDY    211       Cursorspalte als Index ins YR
58940 LDA   (209),Y    Zeichen vom Bildschirm lesen
58942 STA    215       vom Bildschirmcode in ASCII umwandeln
58944 AND    #63
58946 ASL    215
58948 BIT    215
58950 BPL   58954      Bit 6 im BSC gesetzt? Nein: weiter bei 58954
58952 ORA   #128       Bit 7 im Zeichencode setzen
58954 BCC   58960      BSC groesser 127 (Revers)? Nein: weiter bei 58960
58956 LDX    212       Flag fuer Anfuehrungszeichen oder Bit 5 im BSC gesetzt?
58958 BNE   58964
58960 BVS   58964      Ja: weiter bei 58964
58962 ORA    #64       Bit 6 im Zeichencode setzen
58964 INC    211       Pointer fuer Cursorspalte erhoehen
58966 JSR   59012      > Anfuehrungszeichen pruefen, evtl. Quote Modus setzen
58969 CPY    200       letztes Zeichen gelesen?
58971 BNE   58996      Nein: weiter bei 58996

58973 LDA     #0
58975 STA    208       Flag fuer "Zeile komplett gelesen" setzen
58977 LDA    #13       Code fuer "RETURN" als Kennzeichnung fuer Zeilenende
58979 LDX    153       aktiver Eingabe-Kanal
58981 CPX     #3       = Bildschirm?
58983 BEQ   58991      Ja: weiter bei 58991
58985 LDX    154       aktiver Ausgabe-Kanal
58987 CPX     #3       = Bildschirm?
58989 BEQ   58994      Ja: weiter bei 58994
58991 JSR   59158      > Zeichen auf Bildschirm ausgeben
58994 LDA    #13       Code fuer "RETURN"
58996 STA    215       in Register fuer letztes Zeichen bringen
58998 PLA              Indexregister vom Stack holen
58999 TAX
59000 PLA
59001 TAY
59002 LDA    215       gelesenes Zeichen gleich dem
59004 CMP   #222       ASCII fuer PI?
59006 BNE   59010      Nein: fertig
59008 LDA   #255       ASCII fuer PI durch BASIC-Token fuer PI ersetzen
59010 CLC
59011 RTS
```

Umschalter fuer Quote-Modus

```
59012 CMP    #34       Code fuer Anfuehrungszeichen?
59014 BNE   59024      Nein: RTS
59016 LDA    212       Flag fuer Quote-Modus invertieren
59018 EOR     #1
59020 STA    212
59022 LDA    #34       Code fuer Anfuehrungszeichen wiederherstellen
59024 RTS
```

Zeichenausgabe abschliessen (mehrere verschiedene Einspruenge)

```
59025 ORA    #64       Bit 6 im Zeichencode setzen
59027 LDX    199       Flag fuer RVS-Modus gesetzt?
59029 BEQ  59033       Nein: weiter bei 59033
59031 ORA   #128       ansonsten Bit 7 (Revers) im Zeichencode setzen
59033 LDX    216       Insertzaehler
59035 BEQ  59039       = 0? Ja: weiter bei 59039
59037 DEC    216       Insertzaehler vermindern
59039 LDX    646       aktueller Farbcode ins XR
59042 JSR  59923       > Zeichen und Farbe ausgeben
59045 JSR  59062       > Cursor weiterbewegen

59048 PLA              YR wiederherstellen
59049 TAY
59050 LDA    216       Insertzaehler
59052 BEQ  59056       = 0? Ja: weiter bei 59056
59054 LSR    212       ansonsten Quote-Modus ausschalten
59056 PLA              XR und Accu wiederherstellen
59057 TAX
59058 PLA
59059 CLC
59060 CLI
59061 RTS
```

Cursor weiterbewegen, bei Bedarf naechste Zeile initialisieren

```
59062 JSR  59571       > zeilenpointer erhoehen, falls Cursor in letzter Spalte
59065 INC    211       Cursorspaltenpointer erhoehen
59067 LDA    213       aktuelle Zeilenlaenge (39 oder 79)
59069 CMP    211       mit Cursorspalte vergleichen
59071 BCS  59136       letzte Spalte ueberschritten? Nein: RTS
59073 CMP    #79       bereits Doppelzeile?
59075 BEQ  59127       Ja: weiter bei 59127
59077 LDA    658       Zeilenuebergang: erfolgte die Ausgabe des Zeichens
59080 BEQ  59085       im Editiermodus? Nein: weiter bei 59085
59082 JMP  59751       > Fortsetzungzeile einfuegen, Rest nach unten verschieben

59085 LDX    214       Cursorzeile
59087 CPX    #25       kleiner 25?
59089 BCC  59098       Ja: weiter bei 59098
59091 JSR  59626       > SCROLL, Bildschirminhalt nach oben verschieben
59094 DEC    214       Cursorzeilenpointer vermindern
59096 LDX    214

59098 ASL    217,X     Zeile (Nummer im XR) als Fortsetzungszeile markieren
59100 LSR    217,X
59102 INX
59103 LDA    217,X     Folgezeile als Startzeile markieren
59105 ORA   #128
59107 STA    217,X
59109 DEX
59110 LDA    213       aktuelle Zeilenlaenge
59112 CLC
59113 ADC    #40       um 40 erhoehen
59115 STA    213

59117 LDA    217,X     Doppelzeile?
59119 BMI  59124       Nein: weiter bei 59124
59121 DEX              Index vermindern
59122 BNE  59117       weiter zugehoerige Startzeile der Doppelzeile suchen
59124 JMP  59888       > Startadresse low, high der Startzeile setzen
```

Bewegung aus Doppelzeile heraus, neue Zeile initialisieren

```
59127 DEC   214      Cursorzeile vermindern
59129 JSR  59516     > Zeile initialisieren
59132 LDA    #0
59134 STA   211      Spaltenpointer ruecksetzen
59136 RTS
```

Rueckschritt in nicht zur aktuellen Cursorzeile gehoerige Zeile

```
59137 LDX   214      Zeilenpointer
59139 BNE  59147     = 0? Nein: weiter bei 59147
59141 STX   211      Spaltenpointer auf erste Spalte setzen
59143 PLA            Ruecksprungadresse vom Stack holen
59144 PLA
59145 BNE  59048     Unbegdingter Sprung zum Abschluss der Zeichenausgabe

59147 DEX
59148 STX   214      Zeilenpointer vermindern
59150 JSR  58732     > Zeilenparameter aktualisieren
59153 LDY   213      Laenge der Zeile, in die Cursor bewegt wurde
59155 STY   211      als Cursorspalte speichern (letzte Spalte der Zeile)
59157 RTS
```

Ausgabe eines Zeichens auf Bildschirm

```
59158 PHA            Register retten
59159 STA   215
59161 TXA
59162 PHA
59163 TYA
59164 PHA
59165 LDA    #0
59167 STA   208      CR-Flag ruecksetzen
59169 LDY   211      Cursorspalte ins YR,
59171 LDA   215      Zeichencode in Accu bringen
59173 BPL  59178     Code kleiner 128? Ja: weiter bei 59178
59175 JMP  59348     > Shiftzeichen bearbeiten

59178 CMP   #13      "RETURN"?
59180 BNE  59185     Nein: weiter bei 59185
59182 JMP  59537     > Zeilenvorschub, Flag ruecksetzen
59185 CMP   #32      Code kleiner 32 (Steuercodes)?
59187 BCC  59205     Ja: weiter bei 59205
59189 CMP   #96      Code kleiner 96?
59191 BCC  59197     Ja: keine Graphikzeichen, weiter bei 59197
59193 AND  #223      Bit 5 loeschen, Umwandlung in Bildschirmcode
59195 BNE  59199     Unbedingter Sprung

59197 AND   #63      Bit 6 und 7 loeschen, Umwandlung in Bildschirmcode
59199 JSR  59012     > auf Anfuehrungszeichen pruefen
59202 JMP  59027     > Zeichenausgabe abschliessen

59205 LDX   216      Insertzaehler
59207 BEQ  59212     =0? Ja: weiter bei 59212
59209 JMP  59031     > Zeichenausgabe abschliessen

59212 CMP   #20      "DELETE"?
59214 BNE  59262     Nein: weiter bei 59262
59216 TYA            Anfangsspalte
59217 BNE  59225     =0? Nein: weiter bei 59225
59219 JSR  59137     > Rueckschritt in nicht zur Cursorzeile gehoerige Zeile
59222 JMP  59251     > Zeichen auf Cursorposition loeschen, Abschluss
```

Ausgabe eines Zeichens auf Bildschirm (Fortsetzung)

```
59225 JSR  59553       > Pruefung, ob Rueckschritt innerhalb von Doppelzeile
59228 DEY              Cursorspaltenpointer vermindern
59229 STY   211        und als neuen Spaltenpointer speichern
59231 JSR  59940       > Color-Pointer errechnen
59234 INY
59235 LDA   (209),Y    Alle Zeichen von Cursorposition bis
59237 DEY              Zeilenende um eine Stelle zurueckschieben
59238 STA   (209),Y
59240 INY
59241 LDA   (243),Y    ebenso die Farbcodes
59243 DEY
59244 STA   (243),Y
59246 INY
59247 CPY   213        Endspalte erreicht?
59249 BNE  59234       Nein: weitermachen ...
59251 LDA   #32
59253 STA   (209),Y    letzte Position loeschen
59255 LDA   646        aktueller Farbcode
59258 STA   (243),Y    ins Color-RAM
59260 BPL  59339       im Normalfall unbedingter Sprung

59262 LDX   212        Insertzaehler
59264 BEQ  59269       = 0? Ja: weiter bei 59269
59266 JMP  59031       > Zeichenausgabe abschliessen

59269 CMP   #18        "REVERS ON"?
59271 BNE  59275       Nein: weiter bei 59275
59273 STA   199        Revers-Flag setzen
59275 CMP   #19        "HOME"?
59277 BNE  59282       Nein: weiter bei 59282
59279 JSR  58726       Cursor in linke obere Ecke setzen
59282 CMP   #29        "CURSOR RIGHT"?
59284 BNE  59309       Nein: weiter bei 59309
59286 INY              Spaltenpointer erhoehen
59287 JSR  59571       > Pruefung, ob Cursor in letzter Spaltenposition
59290 STY   211        neuer Spaltenpointer
59292 DEY
59293 CPY   213        Uebergang in neue Zeile?
59295 BCC  59306       Nein: Abschluss Zeichenausgabe
59297 DEC   214        Zeilenpointer vermindern
59299 JSR  59516       > neue Zeile initialisieren
59302 LDY   #0         Spaltenpointer
59304 STY   211        auf erste Spalte setzen
59306 JMP  59048       > Abschluss der Zeichenausgabe

59309 CMP   #17        "CURSOR DOWN"?
59311 BNE  59342       Nein: weiter bei 59342
59313 CLC
59314 TYA
59315 ADC   #40        Zeilenlaenge zum Spaltenpointer addieren
59317 TAY
59318 INC   214        Zeilenpointer erhoehen
59320 CMP   213        neue Zeile erreicht?
59322 BCC  59304
59324 BEQ  59304       Nein: Schritt innerhalb von Doppelzeile, weiter bei 59304
59326 DEC   214        Zeilenpointer vermindern
59328 SBC   #40        Spaltenpointer um 40 vermindern (Carry bereits gesetzt)
59330 BCC  59336       bereits genug 40er subtrahiert? Ja: weiter bei 59336
59332 STA   211        Cursorspaltenpointer setzen
59334 BNE  59328       und ein weiteres Mal ...
```

Ausgabe eines Zeichens auf Bildschirm (Fortsetzung)

```
59336 JSR  59516       > neue Zeile initialisieren
59339 JMP  59048       > Abschluss der Zeichenausgabe

59342 JSR  59595       > Controlcodes fuer Farbe pruefen
59345 JMP  60484       > weitere Zeichencodes bearbeiten
```

Bearbeitung der Shiftzeichen

```
59348 AND   #127       Bit 7 loeschen
59350 CMP   #127       war es Code fuer PI?
59352 BNE  59356       Nein: weiter bei 59356
59354 LDA    #94       durch Bildschirmcode fuer PI ersetzen
59356 CMP    #32       Code im Bereich von 128 bis 159 (Steuerzeichen)?
59358 BCC  59363       Ja: weiter bei 59363
59360 JMP  59025       > Zeichenausgabe abschliessen

59363 CMP    #13       Code fuer "Shift-RETURN"
59365 BNE  59370       Nein: weiter bei 59370
59367 JMP  59537       > Zeilenvorschub, Flags ruecksetzen

59370 LDX    212       Quote-Modus eingeschaltet?
59372 BNE  59437       Ja: weiter bei 59437
59374 CMP    #20       Code fuer "INSERT"?
59376 BNE  59433       Nein: weiter bei 59433
59378 LDY    213       Zeilenlaenge
59380 LDA   (209),Y    Zeichen lesen
59382 CMP    #32       gleich "SPACE"?
59384 BNE  59390       Nein: weiter bei 59390
59386 CPY    211       Cursorposition erreicht?
59388 BNE  59397       Nein :weiter bei 59397
59390 CPY    #79       Zeilenlaenge bereits 79 (Doppelzeile)?
59392 BEQ  59430       Ja: Zeichenausgabe abschliessen
59394 JSR  59749       > Fortsetzungszeile anfuegen
59397 LDY    213       Zeilenlaenge
59399 JSR  59940       > Color-Pointer berechnen
59402 DEY              saemtliche Zeichen vom Zeilenende bis zur Cursorposition
59403 LDA   (209),Y    um eine Position nach rechts verschieben
59405 INY
59406 STA   (209),Y
59408 DEY
59409 LDA   (243),Y    Farbcodes ebenso
59411 INY
59412 STA   (243),Y
59414 DEY
59415 CPY    211       Cursorposition erreicht?
59417 BNE  59402       Nein: weitermachen ...
59419 LDA    #32       Cursorposition loeschen
59421 STA   (209),Y
59423 LDA    646       aktuellen Farbcode
59426 STA   (243),Y    ins Color-RAM
59428 INC    216       Insertzaehler erhoehen
59430 JMP  59048       > Abschluss der Zeichenausgabe

59433 LDX    216       Insertzaehler
59435 BEQ  59442       = 0? Ja: weiter bei 59442
59437 ORA    #64       Bit 6 setzen
59439 JMP  59031       > Zeichen ausgeben

59442 CMP    #17       Code fuer "CURSOR UP"?
59444 BNE  59468       Nein: weiter bei 59468
```

Ausgabe eines Zeichens auf Bildschirm (Fortsetzung)

```
59446 LDX    214       Cursorzeile
59448 BEQ  59505       = 0? Ja: Zeichenausgabe abschliessen, weiter bei 59505
59450 DEC    214       Cursorzeile vermindern
59452 LDA    211       Cursorspalte
59454 SEC              um Zeilenlaenge vermindern
59455 SBC    #40
59457 BCC  59463       Cursorbewegung in Doppelzeile? Nein: weiter bei 59463
59459 STA    211       sonst neue Cursorspalte speichern
59461 BPL  59505       Unbedingter Sprung zum Abschluss der Zeichenausgabe

59463 JSR  58732       > Zeilenparameter aktualisieren
59466 BNE  59505       Unbedingter Sprung zum Abschluss der Zeichenausgabe

59468 CMP    #18       Code fuer "REVERS OFF"?
59470 BNE  59476       Nein: weiter bei 59476
59472 LDA     #0
59474 STA    199       Revers-Flag loeschen
59476 CMP    #29       Code fuer "CURSOR LEFT"?
59478 BNE  59498       Nein: weiter bei 59498
59480 TYA              momentane Spalte = erste Spalte?
59481 BEQ  59492       Ja: weiter bei 59492
59483 JSR  59553       > wenn Rueckschritt in Startzeile, Cursorzeile vermindern
59486 DEY              Spaltenzaehler vermindern
59487 STY    211       und als neuen Spaltenpointer speichern
59489 JMP  59048       > Abschluss der Zeichenausgabe

59492 JSR  59137       > Rueckschritt in nicht zur Cursorzeile gehoerige Zeile
59495 JMP  59048       > Abschluss der Zeichenausgabe

59498 CMP    #19       Code fuer "CLEAR SCREEN"
59500 BNE  59508       Nein: weiter bei 59508
59502 JSR  58692       > Bildschirm loeschen
59505 JMP  59048       > Abschluss der Zeichenausgabe

59508 ORA   #128       Bit 7 setzen
59510 JSR  59595       > Controlcodes fuer Farbe pruefen
59513 JMP  60495       > weitere Zeichencodes abfragen
```

Neue Zeile initialisieren

```
59516 LSR    201       Flag fuer Cursorzeilenwechsel (vgl. 58911 ff)
59518 LDX    214       Cursorzeilenpointer
59520 INX              um eins erhoehen
59521 CPX    #25       letzte Zeile erreicht?
59523 BNE  59528       Nein: weiter bei 59528
59525 JSR  59626       > SCROLL, Bildschirminhalt nach oben verschieben
59528 LDA    217,X     Fortsetzungszeile?
59530 BPL  59520       Ja: noch einmal scrollen
59532 STX    214       neue Cursorzeile setzen
59534 JMP  58732       > Zeilenparameter aktualisieren
```

Zeilenvorschub, Flags loeschen (Carriage Return)

```
59537 LDX     #0
59539 STX    216       Insert-Zaehler loeschen
59541 STX    199       Revers-Flag,
59543 STX    212       Quote-Modus ruecksetzen
59545 STX    211       Cursor in erste Spalte setzen
59547 JSR  59516       > neue Zeile initialisieren
59550 JMP  59048       > Abschluss der Zeichenausgabe
```

Pruefung, ob Rueckschritt von zweiter in erste Zeile einer Doppelzeile

```
59553 LDX     #2       maximale Anzahl zusammengehoriger Zeilen
59555 LDA     #0
59557 CMP    211       Cursorspalte = (Accu)?
59559 BEQ  59568       Ja: weiter bei 59568
59561 CLC
59562 ADC    #40       Zeilenlaenge addieren
59564 DEX
59565 BNE  59557
59567 RTS

59568 DEC    214       Cursorzeilenpointer vermindern
59570 RTS
```

Pruefung, ob Cursor in letzter Spalte (beim Bewegen des Cursors nach rechts)

```
59571 LDX     #2       maximale Anzahl zusammenhaengender Zeilen
59573 LDA    #39       Laenge einer Einfachzeile
59575 CMP    211       Cursorspalte mit Accu vergleichen
59577 BEQ  59586       Gleich: weiter bei 59586
59579 CLC
59580 ADC    #40       Zeilenlaenge addieren
59582 DEX
59583 BNE  59575       und noch ein zweites Mal pruefen
59585 RTS

59586 LDX    214       Cursorzeilenpointer
59588 CPX    #25       = 25?
59590 BEQ  59594       Ja: RTS
59592 INC    214       sonst Cursorzeilenpointer erhoehen
59594 RTS
```

Controlcodes fuer Farbe pruefen

```
59595 LDX    #15       Anzahl moeglicher Farben - 1
59597 CMP  59610,X     Steuerzeichen mit Wert in Tabelle vergleichen
59600 BEQ  59606       Gleich: Farbe setzen, weiter bei 59606
59602 DEX
59603 BPL  59597
59605 RTS

59606 STX    646       Farbe setzen (in Adresse fuer aktuellen Farbcode)
59609 RTS

59610 144   5  28 159 156  30  31 158   zugehoerige Tabelle der Farbcodes
59618 129 149 150 151 152 153 154 155
```

SCROLL, Bildschirminhalt nach oben verschieben

```
59626 LDA    172       (172,...,175) auf Stack retten
59628 PHA
59629 LDA    173
59631 PHA
59632 LDA    174
59634 PHA
59635 LDA    175
59637 PHA
59638 LDX   #255       XR als Zeilenzaehler initialisieren
59640 DEC    214       Cursorzeilenpointer vermindern
59642 DEC    201       Cursorzeilenpointer (Aufruf von CHRIN, vgl. 58907 ff)
59644 DEC    677       und Nummer der Fortsetzungszeile um eins vermindern
59647 INX
```

SCROLL, Bildschirminhalt nach oben verschieben (Fortsetzung)

```
59648 JSR   59888      > Pointer auf Zeilenanfang setzen
59651 CPX     #24      saemtliche Zeilen verschoben?
59653 BCS   59667      Ja: weiter bei 59667
59655 LDA   60657,X    Tabelle der LSBs der Zeilenanfaenge des Bildschirms
59658 STA     172      als Pointer low setzen
59660 LDA     218,X    Tabelle fuer MSBs und Doppelzeilenkennzeichnung
59662 JSR   59848      > Pointer setzen, Zeile in vorherige Zeile kopieren
59665 BMI   59647      Unbedingter Sprung

59667 JSR   59903      > letzte Bildschirmzeile loeschen
59670 LDX      #0      MSBs und Doppelzeilenkennzeichnung verschieben
59672 LDA     217,X
59674 AND    #127
59676 LDY     218,X
59678 BPL   59682
59680 ORA    #128
59682 STA     217,X
59684 INX
59685 CPX     #24
59687 BNE   59672
59689 LDA     241      neue Zeile am unteren Ende
59691 ORA    #128      als Einfachzeile kennzeichnen
59693 STA     241
59695 LDA     217      Ist oberste Zeile eine Fortsetzungszeile?
59697 BPL   59638      Ja: noch einmal scrollen
59699 INC     214      Cursorzeilenpointer erhoehen
59701 INC     677      Nummer der Fortsetzungzeile um eins erhoehen
59704 LDA    #127
59706 STA   56320      Row-Select fuer Tastaturabfrage
59709 LDA   56321      Column-Select fuer Tastaturabfrage
59712 CMP    #251      Control-Taste gedrueckt?
59714 PHP
59715 LDA    #127
59717 STA   56320
59720 PLP
59721 BNE   59734      Nein: weiter bei 59734
59723 LDY      #0      Verzoegerungsschleife
59725 NOP
59726 DEX
59727 BNE   59725
59729 DEY
59730 BNE   59725
59732 STY     198      Tastaturpuffer loeschen
59734 LDX     214      Cursorzeilenpointer ins XR

59736 PLA              (172,...,175) wiederherstellen
59737 STA     175
59739 PLA
59740 STA     174
59742 PLA
59743 STA     173
59745 PLA
59746 STA     172
59748 RTS
```

Fortsetzungszeile anfuegen

```
59749 LDX    214      Cursorzeilenpointer
59751 INX
59752 LDA    217,X    gehoert Zeile unter Cursorzeile bereits zur Cursorzeile
59754 BPL   59751     Ja: naechste nichtzugehoerige Zeile suchen
59756 STX    677      Nummer der anzufuegenden Zeile
59759 CPX    #24      kleiner oder gleich 24 (also noch im Bildschirm)?
59761 BEQ   59777
59763 BCC   59777     Ja: weiter bei 59777
59765 JSR   59626     > SCROLL, Bildschirminhalt nach oben verschieben
59768 LDX    677      Nummer der anzufuegenden Zeile
59771 DEX             um eins vermindern
59772 DEC    214      ebenso Cursorzeilenpointer
59774 JMP   59098     > Fortsetzungszeile initialisieren

59777 LDA    172      (172,...,175) auf Stack retten
59779 PHA
59780 LDA    173
59782 PHA
59783 LDA    174
59785 PHA
59786 LDA    175
59788 PHA
59789 LDX    #25      XR als Zeilenzaehler initialisieren
59791 DEX
59792 JSR   59888     > Zeilenpointer (209/210) setzen
59795 CPX    677      bereits alle Zeilen bis Cursorzeile verschoben?
59798 BCC   59814
59800 BEQ   59814     Ja: weiter bei 59814
59802 LDA   60655,X   Tabelle der LSBs der Zeilenanfaenge
59805 STA    172      als Pointer low setzen
59807 LDA    216,X    Tabelle der MSBs und Doppelzeilenkennzeichnung
59809 JSR   59848     > Zeileninhalt um eine Zeile nach unten kopieren
59812 BMI   59791     Unbedingter Sprung

59814 JSR   59903     > eingefuegte Zeile loeschen
59817 LDX    #23      Tabelle der MSBs verschieben
59819 CPX    677      bereits alles verschoben?
59822 BCC   59839     Ja: weiter bei 59839
59824 LDA    218,X    Tabelle der MSBs und Doppelzeilenkennzeichnug
59826 AND   #127      nach 'unten' verschieben
59828 LDY    217,X
59830 BPL   59834
59832 ORA   #128
59834 STA    218,X
59836 DEX
59837 BNE   59819     weiter bei 59819
59839 LDX    677      Nummer der eingefuegten Zeile
59842 JSR   59098     > Zeile als Doppelzeile markieren und initialisieren
59845 JMP   59736     > (172,...,175) wiederherstellen
```

Zeile in andere Zeile kopieren (fuer SCROLL und 'Zeile einfuegen')

```
59848 AND    #3       Adresse high der Zeile
59850 ORA   648       'berechnen'
59853 STA   173       und als Pointer high speichern
59855 JSR 59872       > Transferpointer in Color-RAM berechnen
59858 LDY   #39
59860 LDA (172),Y     Zeileninhalt Zeichen fuer Zeichen
59862 STA (209),Y     in neue Zeile verschieben
59864 LDA (174),Y     ebenso die Farbcodes
59866 STA (243),Y
59868 DEY             alle 40 Zeichen verschoben?
59869 BPL 59860       Nein: weitermachen ...
59871 RTS
```

Pointer in Color-RAM berechnen

```
59872 JSR 59940       > Pointer in Color-RAM berechnen
59875 LDA   172       Transferpointer fuer Quellzeile in Colorpointer
59877 STA   174       fuer Quellbereich umwandeln
59879 LDA   173
59881 AND    #3       Bits 0 und 1 isolieren
59883 ORA  #216       216 * 256 = 55296, Startadresse des Color-Nybble-RAMs
59885 STA   175
59887 RTS
```

Pointer in (209/210) auf Zeile (Nummer im XR) setzen

```
59888 LDA 60656,X     Tabelle der LSBs der Zeilenanfaenge
59891 STA   209       Pointer low
59893 LDA   217,X     Tabelle der MSBs und Doppelzeilenkennzeichnung
59895 AND    #3       Bits 0 und 1 isolieren
59897 ORA   648       Startposition high des Bildschirm-RAMs
59900 STA   210       ergibt Pointer high
59902 RTS
```

Zeile loeschen

```
59903 LDY   #39       Laenge der Zeile - 1
59905 JSR 59888       > Zeilenpointer (209/210) setzen
59908 JSR 59940       > Pointer in Color-RAM berechnen
59911 LDA   #32       Leercode
59913 STA (209),Y     in Bildschirm-RAM bringen
59915 JSR 58586       > Background-Color in Color-RAM bringen
59918 NOP
59919 DEY             alle 40 Zeichen geloescht?
59920 BPL 59911       Nein: weitermachen ...
59922 RTS
```

Zeichen (Accu) und Farbcode (im XR) auf Cursorposition speichern

```
59923 TAY             Accu merken
59924 LDA    #2
59926 STA   205       Zaehler fuer Blinkdauer des Cursors (fuer REPEAT)
59928 JSR 59940       > Pointer in Color-RAM berechnen
59931 TYA
59932 LDY   211       Pointer auf Cursorspalte
59934 STA (209),Y     Zeichen speichern
59936 TXA
59937 STA (243),Y     Farbwert speichern
59939 RTS
```

Pointer in Color-RAM aus Pointer in Bildschirm-RAM berechnen

```
59940 LDA    209       Position des Cursors im Bildschirm-RAM
59942 STA    243       in Position des Cursors im Color-Nybble-RAM
59944 LDA    210       umwandeln
59946 AND     #3
59948 ORA   #216
59950 STA    244
59952 RTS
```

IRQ-Routine, wird im Normalfall alle 60stel Sekunden aufgerufen

```
59953 JSR  65514       > UDTIM, Uhrzeit um eine 60stel Sekunde weitersetzen
59956 LDA    204       Cursor eingeschaltet?
59958 BNE  60001       Nein: Cursor-Behandlung ueberspringen
59960 DEC    205       Zaehler fuer Blinkdauer des Cursors vermindern
59962 BNE  60001       Zeit vorbei? Nein: weiter 60001
59964 LDA    #20
59966 STA    205       Zaehler wieder neu initialisieren
59968 LDY    211       Cursorspalte in YR
59970 LSR    207       Flag fuer Cursor (hell/dunkel)
59972 LDX    647       Farbcode unter Cursor
59975 LDA   (209),Y    Zeichen unter Cursor vom Bildschirm lesen
59977 BCS  59996       war Cursor dunkel? Nein: weiter bei 59996
59979 INC    207       Flag fuer 'Cursor hell' setzen
59981 STA    206       Zeichen unter Cursor merken
59983 JSR  59940       > Pointer in Color-RAM berechnen
59986 LDA   (243),Y    Farbcode unter Cursor
59988 STA    647       merken
59991 LDX    646       momentanen Farbcode (Cursor- und PRINT-Farbe) ins XR
59994 LDA    206       Zeichen unter Cursor
59996 EOR   #128       invertieren
59998 JSR  59932       und zusammen mit Farbwert auf Bildschirm bringen
60001 LDA      1
60003 AND    #16       Recordertaste gedrueckt?
60005 BEQ  60017       Ja: weiter bei 60017
60007 LDY     #0
60009 STY    192       Flag fuer Motorkontrolle loeschen
60011 LDA      1
60013 ORA    #32       Motor ausschalten
60015 BNE  60025       Unbedingter Sprung

60017 LDA    192       Flag fuer Motorkontrolle gesetzt?
60019 BNE  60027       Nein: weiter bei 60027
60021 LDA      1
60023 AND    #31       Motor einschalten
60025 STA      1
60027 JSR  60039       > SCNKEY, Tastaturabfrage
60030 LDA  56333

60033 PLA              Register wiederherstellen
60034 TAY
60035 PLA
60036 TAX
60037 PLA
60038 RTI
```

SCNKEY, Tastaturabfrage

```
60039 LDA     #0       Flag fuer Shift (Bit 0), CBM-Taste (Bit 1)
60041 STA    653       und Control (Bit 2) loeschen
60044 LDY    #64
60046 STY    203       Tastaturcode mit 64 (keine Taste gedrueckt) vorbelegen
60048 STA  56320       Null nach 59320 (fragt alle Tasten ab)
60051 LDX  56321       Column-Select
60054 CPX   #255       = 255 (keine der Tasten gedrueckt)?
60056 BEQ  60155       Ja: weiter bei 60198
60058 TAY              YR := 0
60059 LDA   #129       (245/246) := 60289
60061 STA    245       (Startadresse der Tastaturdecodierungstabelle)
60063 LDA   #235
60065 STA    246
60067 LDA   #254       erste Spalte abfragen (Bit 0 geloescht)
60069 STA  56320       nach Row-Select bringen
60072 LDX     #8       Zaehler fuer 8 Spalten
60074 PHA              (Wert fuer Row-Select) merken
60075 LDA  56321       Dekoder-Ausgang
60078 CMP  56321       nochmalige Abfrage zum Entprellen
60081 BNE  60075
60083 LSR              Bit herausshiften
60084 BCS  60108       gesetzt (Taste nicht gedrueckt)? Ja: weiter bei 60108
60086 PHA              Wert merken
60087 LDA  (245),Y     Wert aus Tabelle holen
60089 CMP     #5       groesser als 4 (keine Kontrolltaste)
60091 BCS  60105       Ja: weiter bei 60105
60093 CMP     #3       oder gleich 3 (RUNSTOP-Taste)?
60095 BEQ  60105       Ja: weiter bei 60105
60097 ORA    653       entsprechendes Flag fuer Shift, CBM-Taste
60100 STA    653       oder Control setzen
60103 BPL  60107       Unbedingter Sprung

60105 STY    203       Wert als Tastaturmatrixcode speichern
60107 PLA              Wert aus Dekoder-Ausgang wiederherstellen
60108 INY
60109 CPY    #65       alle 64 Tasten abgefragt?
60111 BCS  60124       Ja: weiter bei 60124
60113 DEX              Zaehler fuer Row-Select vermindern
60114 BNE  60083
60116 SEC              Carry setzen (fuer ROL)
60117 PLA              Wert fuer Row-Select wiederherstellen
60118 ROL              und geloeschtes Bit weiter nach links verschieben
60119 STA  56320       fuer Column-Select neu speichern
60122 BNE  60072       Unbedingter Sprung

60124 PLA              Stack normalisieren
60125 JMP   (655)      Normalwert des Vektors (655/656): 60232

60128 LDY    203       Tastaturmatrixcode
60130 LDA  (245),Y     Wert aus Dekodierungstabelle lesen
60132 TAX              im XR merken
60133 CPY    197       mit Wert aus vorherigem Aufruf von SCNKEY vergleichen
60135 BEQ  60144       identisch? Ja: weiter bei 60144
60137 LDY    #16       neue Taste;
60139 STY    652       REPEAT-Zaehler neu initialisieren
60142 BNE  60198       Unbedingter Sprung

60144 AND   #127       Bit 7 loeschen
60146 BIT    650       Flag fuer REPEAT
60149 BMI  60173       Bit 7 gesetzt (Repeat aller Tasten)? Ja: weiter bei 60173
60151 BVS  60226       Bit 6 gesetzt (kein Repeat)? Ja: weiter bei 60226
```

SCNKEY, Tastaturabfrage (Fortsetzung)

```
60153 CMP   #127      kein Zeichen?
60155 BEQ  60198      Ja: weiter bei 60198
60157 CMP   #20       Code fuer "DELETE"?
60159 BEQ  60173      Ja: weiter bei 60173
60161 CMP   #32       Code fuer "SPACE"?
60163 BEQ  60173      Ja: weiter bei 60173
60165 CMP   #29       Code fuer "CURSOR RIGHT"?
60167 BEQ  60173      Ja: weiter bei 60173
60169 CMP   #17       Code fuer "CURSOR DOWN"?
60171 BNE  60226      Nein: weiter bei 60226
60173 LDY   652       Zaehler fuer Anspruchszeit fuer REPEAT abgelaufen?
60176 BEQ  60183      Ja: weiter bei 60183
60178 DEC   652       Zaehler weiter herabzaehlen
60181 BNE  60226      = 0? Nein: weiter bei 60226
60183 DEC   651       Zaehler fuer Wiederholungsgeschwindigkeit herabzaehlen
60186 BNE  60226      = 0? Nein: weiter bei 60226
60188 LDY    #4       Zaehler fuer
60190 STY   651       Tastenwiederholungsgeschwindigkeit neu initialisieren
60193 LDY   198       Anzahl Zeichen im Tastaturpuffer
60195 DEY
60196 BPL  60226      ungleich Null? Ja: weiter bei 60226
60198 LDY   203       Tastaturmatrixcodes umspeichern
60200 STY   197
60202 LDY   653       ebenso die Flags fuer Shift, CBM-Taste und Control
60205 STY   654
60208 CPX  #255       Zeichen aus Dekodierungstabelle = 255?
60210 BEQ  60226      Ja: kein Zeichen, weiter bei 60226
60212 TXA
60213 LDX   198       Anzahl Zeichen im Tastaturpuffer
60215 CPX   649       mit maximaler Anzahl Zeichen vergleichen
60218 BCS  60226      Groesser oder gleich? Ja: nicht abspeichern
60220 STA   631,X     sonst Zeichen am Ende des Tastaturpuffers anfuegen
60223 INX
60224 STX   198       Anzahl Zeichen um eins erhoehen
60226 LDA  #127       Normalwert
60228 STA  56320      in Row-Select einschreiben
60231 RTS
```

Umschaltung von Graphik und Kleinschrift ueber Commodore-Taste

```
60232 LDA   653       Flag fuer Shift, Commodore-Taste und Control
60235 CMP    #3       Shift und Commodore-Taste gedrueckt?
60237 BNE  60260      Nein: weiter bei 60260
60239 CMP   654       waren beide Tasten schon beim letzen Aufruf gedrueckt?
60242 BEQ  60226      Ja: Abschluss SCNKEY, weiter bei 60226
60244 LDA   657       Umschaltung blockiert (vgl. 60510 ff)?
60247 BMI  60278      Ja: weiter bei 60128
60249 LDA  53272
60252 EOR    #2
60254 STA  53272      Umschaltung fuer Graphik und Kleinschrift
60257 JMP  60278      > weiter bei 60128
60260 ASL
60261 CMP    #8       war die Control-Taste gedrueckt?
60263 BCC  60267      Nein: weiter bei 60267
60265 LDA    #6       Offset fuer Tastaturdecodierungspointer 60536
60267 TAX
60268 LDA  60281,X    Pointer (245/246) auf Decodierungstabelle setzen
60271 STA   245
60273 LDA  60282,X
60276 STA   246
60278 JMP  60128      > Fortsetzung der Tastaturabfrage
```

Pointer auf Tastaturdecodierungstabellen

```
60281 161 235   60289     Tabelle fuer einzelne Tasten
60283 194 235   60354     Tabelle fuer Tasten zusammen mit Shift
60285   3 236   60419     Tabelle fuer Tasten zusammen mit Commodore-Taste
60287 120 236   60536     Tabelle fuer Tasten zusammen mit Control
```

Tabellen fuer Tastaturdecodierung:

```
60289  20  13  29 136 133 134 135  17  51  87     einzelne Tasten
60299  65  52  90  83  69   1  53  82  68  54
60309  67  70  84  88  55  89  71  56  66  72
60319  85  86  57  73  74  48  77  75  79  78
60329  43  80  76  45  46  58  64  44  92  42
60339  59  19   1  61  94  47  49  95   4  50
60349  32   2  81   3 255

60354 148 141 157 140 137 138 139 145  35 215     zusammen mit Shift
60364 193  36 218 211 197   1  37 210 196  38
60374 195 198 212 216  39 217 199  40 194 200
60384 213 214  41 201 202  48 205 203 207 206
60394 219 208 204 221  62  91 186  60 169 192
60404  93 147   1  61 222  63  33  95   4  35
60414 160   2 209 131 255

60419 148 141 157 140 137 138 139 209 150 179     zusammen mit Commodore-Taste
60429 176 151 173 174 177   1 152 178 172 153
60439 188 187 163 189 154 183 165 155 191 180
60449 184 188  41 162 181  48 167 161 185 170
60459 166 175 182 220  62  91 164  60 168 223
60469  93 147   1  61 222  63 129  95   4 149
60479 160   2 171 131 255
```

weitere Zeichencodes abfragen (siehe 59345 und 59513)

```
60484 CMP    #14      Code fuer 'Gross- und Kleinschrift'?
60486 BNE  60495      Nein: weiter bei 60495
60488 LDA  53272
60491 ORA     #2      Bit 1 setzen
60493 BNE  60504      Unbedingter Sprung

60495 CMP   #142      Code fuer 'Grossschrift und Graphik'?
60497 BNE  60510      Nein: weiter bei 60510
60499 LDA  53272
60502 AND   #253      Bit 1 loeschen
60504 STA  53272
60507 JMP  59048      > Ausgabe abschliessen

60510 CMP     #8      Code fuer 'Blockierung der Shift- und Commodore-Taste'?
60512 BNE  60521      Nein: weiter bei 60521
60514 LDA   #128      Bit 7
60516 ORA    657      in 657 setzen
60519 BMI  60530      Unbedingter Sprung

60521 CMP     #9      Code fuer 'Freigabe der Shift- und Commodore-Taste'?
60523 BNE  60507      Nein: keine weiteren Codes mehr, Ausgabe abschliessen
60525 LDA   #127      Bit 7
60527 AND    657      in 657 loeschen
60530 STA    657
60533 JMP  59048      > Ausgabe abschliessen
```

Tabelle fuer Tastaturdekodierung

```
60536 255 255 255 255 255 255 255 255  28  23     zusammen mit Control-Taste
60546   1 159  26  19   5 255 156  18   4  30
60556   3   6  20  24  31  25   7 158   2   8
60566  21  22  18   9  10 146  13  11  15  14
60576 255  16  12 255 255  27   0 255  28 255
60586  29 255 255  31  30 255 144   6 255   5
60596 255 255  17 255 255
```

Daten fuer Video-Controller

```
60601   0   0   0   0   0   0   0   0   0   0
60611   0   0   0   0   0   0   0 155  55   0
60621   0   0   8   0  20  15   0   0   0   0
60631   0   0  14   6   1   2   3   4   0   1
60641   2   3   4   5   6   7
```

Text fuer RUNSTOP-Taste

```
60646  76  79  65  68  13  82  85  78  13        "LOAD" CR "RUN" CR
```

Tabelle der LSBs der Bildschirmanfaenge

```
60656   0  40  80 120 160 200 240  24  64 104
60666 144 184 224   8  48  88 128 168 208 248
60676  32  72 112 152 192
```

Routinen zur Bedienung des seriellen Bus

TALK und LISTEN

```
60681 ORA    #64       Bit 6 setzen, TALK
60683 BIT    ...
60684 ORA    #32       Bit 5 setzen, LISTEN
60686 JSR   61604      > Warten, bis RS-232-Datenuebertragung beendet ist
60689 PHA              Accu merken
60690 BIT    148       Zeichen im Puffer?
60692 BPL   60704      Nein: weiter bei 60704
60694 SEC
60695 ROR    163       Flag fuer EOI setzen
60697 JSR   60736      > Byte ausgeben, EOI
60700 LSR    148       Flag fuer 'Zeichen im Puffer' loeschen
60702 LSR    163       Flag fuer EOI loeschen
60704 PLA              Accu wiederherstellen
60705 STA    149       und in Puffer bringen
60707 SEI
60708 JSR   61079      > Serial Data := low
60711 CMP    #63       Accu kann nach Aufruf von 61079 nicht gleich 63 sein
60713 BNE   60718
60715 JSR   61061      > Clock Out := low
60718 LDA   56576      ATN := high
60721 ORA     #8
60723 STA   56576
60726 SEI
60727 JSR   61070      > Clock Out := high
60730 JSR   61079      > Serial Data := low
60733 JSR   61107      > Verzoegerungsschleife
```

Byte auf Bus ausgeben

```
60736 SEI
60737 JSR   61079       > Serial Data := low
60740 JSR   61097       > Serial Data abfragen
60743 BCS   60845       =1? Ja: Bit 7 (Device not present) im Status setzen
60745 JSR   61061       > Clock Out := low
60748 BIT     163       Flag fuer EOI gesetzt?
60750 BPL   60762       Nein: weiter bei 60762
60752 JSR   61097       > Datenbit vom Bus holen
60755 BCC   60752       =1? Nein: warten ... (warten, bis Listener bereit)
60757 JSR   61097       > Datenbit vom Bus holen
60760 BCS   60757       =0? Nein: warten ... (Timeout fuer EOI erzeugen)
60762 JSR   61097       > Datenbit vom Bus holen
60765 BCC   60762       =1? Nein: warten ... (warten, bis bereit fuer Daten)
60767 JSR   61070       > Clock Out := high
60770 LDA      #8
60772 STA     165       Zaehler fuer 8 Bits setzen
60774 LDA   56576       Port A lesen
60777 CMP   56576
60780 BNE   60774
60782 ASL               Listener bereit?
60783 BCC   60848       Nein: Fehler, weiter bei 60848
60785 ROR     149       Bit aus Ausgaberegister herausshiften
60787 BCS   60794       Ist Bit = 1? Ja: weiter bei 60794
60789 JSR   61088       > Serial Data := high
60792 BNE   60797       Unbedingter Sprung

60794 JSR   61079       > Serial Data := low
60797 JSR   61061       > Clock Out := low (data valid)
60800 NOP
60801 NOP
60802 NOP
60803 NOP
60804 LDA   56576
60807 AND    #223       Serial Data := low
60809 ORA     #16       Clock Out := high
60811 STA   56576
60814 DEC     165       Bit-Zaehler vermindern
60816 BNE   60774       alle Bits ausgegeben? Nein: weitermachen ...
60818 LDA      #4
60820 STA   56327       Timer B fuer Wartezeit setzen
60823 LDA     #25
60825 STA   56335       Modus fuer Timer B setzen
60828 LDA   56333
60831 LDA   56333
60834 AND      #2       Bit 1 gesetzt (Timeout Timer B)?
60836 BNE   60848       Ja: Statusbits fuer Zeitfehler setzen
60838 JSR   61097       > Datenbit vom Bus holen
60841 BCS   60831       =0? Nein: warten ... (bis Listener Empfang quittiert hat)
60843 CLI
60844 RTS
```

Fehlerbehandlung

```
60845 LDA    #128       Bit 7 fuer 'Device not present'
60847 BIT     ...
60848 LDA      #3       Bits 0 und 1 fuer Zeitfehler
60850 JSR   65052       > Status setzen
60853 CLI
60854 CLC
60855 BCC   60931       weiter bei 60931
```

SECOND, Ausgabe Sekundaeradresse nach LISTEN

```
60857 STA   149     Sekundaeradresse in Ausgaberegister
60859 JSR  60726    > Sekundaeradresse ausgeben
60862 LDA  56576    ATN := low (Kontroll-Modus beenden)
60865 AND  #247
60867 STA  56576
60870 RTS
```

TKSA, Ausgabe Sekundaeradresse nach TALK

```
60871 STA   149     Sekundaeradresse in Ausgaberegister
60873 JSR  60726    > Sekundaeradresse ausgeben
60876 SEI
60877 JSR  61088    > Serial Data := high
60880 JSR  60862    > ATN := low
60883 JSR  61061    > Clock Out := low (Kontroll-Modus beenden)
60886 JSR  61097    > Clock In = 0?
60889 BMI  60886    Nein: warten ...
60891 CLI
60892 RTS
```

CIOUT, Zeichen ausgeben (Pufferung fuer EOI)

```
60893 BIT   148     gepuffertes Zeichen vorhanden?
60895 BMI  60902    Ja: weiter bei 60902
60897 SEC
60898 ROR   148     Flag fuer 'Zeichen im Puffer' setzen
60900 BNE  60907    Unbedingter Sprung

60902 PHA
60903 JSR  60736    > Byte im Ausgaberegister auf Bus ausgeben
60906 PLA
60907 STA   149     Zeichen in Puffer bringen
60909 CLC
60910 RTS
```

UNTALK und UNLISTEN

```
60911 SEI
60912 JSR  61070    > Clock Out := high
60915 LDA  56576    ATN := high
60918 ORA    #8
60920 STA  56576
60923 LDA   #95     Kennzeichnung fuer UNTALK
60925 BIT   ...
60926 LDA   #63     Kennzeichnung fuer UNLISTEN
60928 JSR  60689    > Ausgabe mit ATN = high
60931 JSR  60862    > ATN := low (Kontroll-Modus beenden)
60934 TXA           XR merken
60935 LDX   #10     Verzoegerungsschleife um ca. 50 Mikrosekunden
60937 DEX
60938 BNE  60937
60940 TAX           XR wiederherstellen
60941 JSR  61061    > Clock Out := low
60944 JMP  61079    > Serial Data := low
```

ACPTR, Zeichen vom Bus holen

```
60947 SEI
60948 LDA     #0
60950 STA    165       (165) := 0 (EOI, falls Timeout)
60952 JSR  61061       > Clock Out := low
60955 JSR  61097       > Clock In = 1?
60958 BPL  60955       Nein: warten ...
60960 LDA     #1
60962 STA  56327       Wartezeit fuer Timer B setzen
60965 LDA    #25
60967 STA  56335       Modus fuer Timer B setzen
60970 JSR  61079       > Serial Data := low (ready for data)
60973 LDA  56333
60976 LDA  56333
60979 AND     #2       Bit 1 (Timeout Timer B) gesetzt?
60981 BNE  60990       Ja: weiter bei 60990
60983 JSR  61097       > Clock In = 0 (Start der Byteuebertragung)?
60986 BMI  60976       Nein: warten ...
60988 BPL  61014       sonst Byte von Bus holen
60990 LDA    165       (165) = 0?
60992 BEQ  60999       Ja: EOI empfangen, weiter bei 60999
60994 LDA     #2       Bit 1 (Timeout/Input)
60996 JMP  60850       > Status setzen, Fehlerbehandlung
60999 JSR  61088       > Serial Data := high (EOI accepted)
61002 JSR  61061       > Clock Out := low
61005 LDA    #64       Bit 6 (End or Identify)
61007 JSR  65052       > Status setzen
61010 INC    165       (165) := 1 (Zeitfehler, falls Timeout)
61012 BNE  60960       weiter bei 60960, Datenbyte nach EOI holen
```

8 Bits holen

```
61014 LDA     #8
61016 STA    165       Zaehler fuer 8 Bit setzen
61018 LDA  56576       Port A lesen
61021 CMP  56576
61024 BNE  61018
61026 ASL              Clock In = 1?
61027 BPL  61018       Nein: warten bis Daten gueltig
61029 ROR    164       Datenbit (von Bit 0 bis Bit 7) in (164) shiften
61031 LDA  56576       Port A lesen
61034 CMP  56576
61037 BNE  61031
61039 ASL              Clock In = 0?
61040 BMI  61031       Nein: warten ...
61042 DEC    165       alle 8 Bits geholt?
61044 BNE  61018       Nein: weitermachen
61046 JSR  61088       > Serial Data := high (not ready for data)
61049 BIT    144       EOI aufgetreten?
61051 BVC  61056       Nein: weiter bei 61056
61053 JSR  60934       > Clock Out := low, Serial Data := low
61056 LDA    164       geholtes Datenbyte in Accu bringen
61058 CLI
61059 CLC
61060 RTS

61061 LDA  56576       'Clock Out' auf low setzen
61064 AND   #239
61066 STA  56576
61069 RTS
```

Routinen zur Behandlung des seriellen Ports (Fortsetzung)

```
61070 LDA  56576       'Clock Out' auf high setzen
61073 ORA    #16
61075 STA  56576
61078 RTS

61079 LDA  56576       'Serial Data Out' auf low setzen
61082 AND   #223
61084 STA  56576
61087 RTS

61088 LDA  56576       'Serial Data Out' auf high setzen
61091 ORA    #32
61093 STA  56576
61096 RTS

61097 LDA  56576       Daten vom Port A lesen
61100 CMP  56576
61103 BNE  61097
61105 ASL              Datenbit in Carry schieben, Clock In in N-Flag
61106 RTS

61107 TXA              Verzoegerungsschleife um ca. eine Millisekunde
61108 LDX   #184
61110 DEX
61111 BNE  61110
61113 TAX
61114 RTS
```

Routinen zur Bedienung der RS-232 Schnittstelle

Uebertragung des naechsten Bits vorbereiten

```
61115 LDA    180       Bitzaehler fuer Senden
61117 BEQ  61190       = 0 (Byte vollstaendig uebertragen)? Ja: weiter bei 61190
61119 BMI  61184       muss Stopbit uebertragen werden? Ja: weiter bei 61184
61121 LSR    182       Bit aus dem Register fuer zu uebertragendes Byte schieben
61123 LDX     #0       Wert 0, falls Datenbit = 0
61125 BCC  61128       Bit geloescht? Ja: weiter bei 61128
61127 DEX              Wert 255, falls Datenbit = 1
61128 TXA
61129 EOR    189       mit Register fuer fuer Paritybit verknuepfen
61131 STA    189       und wieder dort abspeichern
61133 DEC    180       Bitzaehler vermindern
61135 BEQ  61143       = 0 (alle Datenbits uebertragen)? Ja: weiter bei 61143
61137 TXA
61138 AND     #4       Bit 2 isolieren (Datenbit fuer Uebertragung)
61140 STA    181       und im Register fuer zu sendendes Bit bringen
61142 RTS

61143 LDA    #32       Maske fuer Bit 5 (Flag fuer Parity)
61145 BIT    660       RS-232 Kommandoregister abfragen
61148 BEQ  61170       keine Uebertragung eines Paritybits? Ja: weiter bei 61170
61150 BMI  61180       Uebertragung eines bestimmten Bits statt des Paritybits?
61152 BVS  61174       Odd Parity? Nein: weiter bei 61174
61154 LDA    189       Paritybit = 1?
61156 BNE  61159       Ja: weiter bei 61159
61158 DEX              Wert 255 fuer Parity
61159 DEC    180       Bitzaehler auf 255 setzen
```

Uebertragung des naechsten Bits vorbereiten (Fortsetzung)

```
61161 LDA    659       RS-232 Kontrollregister
61164 BPL  61137       zwei Stoppbits? Nein: Ausgabe des Paritybits vorbereiten
61166 DEC    180       Bitzaehler auf 254 setzen
61168 BNE  61137       Unbedingter Sprung

61170 INC    180       Bitzaehler erhoehen, da keine Paritybit-Ausgabe
61172 BNE  61158       Unbedingter Sprung zur Bestiimmung der Anzahl Stoppbits

61174 LDA    189       Paritybit
61176 BEQ  61159       = 0? Ja: Wert 0 ausgeben
61178 BNE  61158       ansonsten Wert 1 ausgeben

61180 BVS  61159       Uebertragung des Wertes 0,
61182 BVC  61158       Uebertragung des Wertes 1 anstatt des Paritybits

61184 INC    180       Bitzaehler erhoehen (ist bei Stoppbit-Ausgabe unter null)
61186 LDX   #255       Wert fuer Stoppbit
61188 BNE  61137       Unbedingter Sprung
```

Uebertragung eines Bytes vorbereiten

```
61190 LDA    660       RS-232 Kommandoregister
61193 LSR              Handshake-Mode
61194 BCC  61203       X-Line? Nein: weiter bei 61203
61196 BIT  56577       Port B (NMI-CIA) abfragen; liegt 'DSR IN' auf high?
61199 BPL  61230       Nein: 'DSR Signal Missing', RS-232 Status setzen
61201 BVC  61233       Liegt 'CTS IN' auf high? Nein: 'CTS Signal Missing'
61203 LDA     #0
61205 STA    189       Register fuer Pruefsumme loeschen
61207 STA    181       Register fuer zu sendendes Bit loeschen (fuer Startbit)
61209 LDX    664       Register fuer Wortlaenge
61212 STX    180       in Zaehler fuer Bits uebertragen
61214 LDY    669       Ist der Zeiger auf zu uebertragendes Byte gleich
61217 CPY    670       dem Zeiger auf die naechste freie Stelle im Puffer?
61220 BEQ  61241       Ja: alle Zeichen uebertragen, Uebertragung abschliessen
61222 LDA  (249),Y     Zeichen aus Sendepuffer holen
61224 STA    182       und in Register fuer zu uebertragendes Byte bringen
61226 INC    669       Zeiger auf zu sendendes Byte erhoehen
61229 RTS

61230 LDA    #64       Bit 6, 'DSR Signal Missing'
61232 BIT    ...
61233 LDA    #16       Bit 4, 'CTS Signal Missing'
61235 ORA    663       mit vorherigem Wert im RS-232 Statusregister verknuepfen
61238 STA    663       und als neuen Wert wieder dort abspeichern
61241 LDA     #1       Bitwert fuer 'TimerA-NMI sperren'
61243 STA  56589       entsprechenden NMI im ICR sperren/freigeben
61246 EOR    673       mit RS-232 Register fuer aktive NMIs verknuepfen,
61249 ORA   #128       Bit 7 setzen
61251 STA    673       und als neuen Wert wieder abspeichern
61254 STA  56589       uebrige aktive NMIs freigeben
61257 RTS
```

Feststellung der Wortlaenge fuer RS-232 Uebertragung (Ergebnis+1 im XR)

```
61258 LDX    #9
61260 LDA    #32      Maskenwert fuer Bit 5
61262 BIT    659      Testen der Bitkonfiguration im RS-232 Kontrollregister
61265 BEQ  61268      Ist Bit 5 gesetzt? Nein: weiter bei 61268
61267 DEX             Zaehler fuer Wortlaenge um eins vermindern
61268 BVC  61272      Ist Bit 6 gesetzt? Nein: RTS
61270 DEX             Zaehler fuer Wortlaenge um zwei vermindern
61271 DEX
61272 RTS
```

empfangenes Bit verarbeiten

```
61273 LDX    169      Startbit?
61275 BNE  61328      Ja: weiter bei 61328
61277 DEC    168      Bitzaehler vermindern
61279 BEQ  61335      = 0 (Byte vollstaendig empfangen)? Ja: weiter bei 61335
61281 BMI  61296      sind (noch) Stoppbits zu empfangen? Ja: weiter bei 61296
61283 LDA    167      empfangenes Bit
61285 EOR    171      mit Register fuer Paritybit verknuepfen
61287 STA    171      und wieder dort abspeichern
61289 LSR    167      empfangenes Bit
61291 ROR    170      in serielles Empfangsregister schieben
61293 RTS

61294 DEC    168      Bitzaehler vermindern
61296 LDA    167      empfangenes Stoppbit
61298 BEQ  61403      = 0? Ja: weiter bei 61403
61300 LDA    659      RS-232 Kontrollregister
61303 ASL             Bit fuer Anzahl Stoppbits
61304 LDA    #1
61306 ADC    168      Carry + 1 + Bitzaehler
61308 BNE  61293      alle Stoppbits empfangen? Nein: RTS
61310 LDA   #144      Bitwert fuer 'FLAG-NMI (Startbit), RS-232 IN freigeben'
61312 STA  56589      ICR setzen
61315 ORA    673
61318 STA    673      und RS-232 Register fuer aktive NMIs setzen
61321 STA    169      Flag fuer Startbit setzen
61323 LDA    #2       Bitwert fuer 'TimerB-NMI sperren'
61325 JMP  61243      > ICR und Register fuer aktive NMIs setzen

61328 LDA    167      Startbit
61330 BNE  61310      = 0? Nein: weiter bei 61310
61332 STA    169      sonst alles in Ordnung, Flag fuer Startbit ruecksetzen
61334 RTS
```

Verarbeitung des empfangenen Bytes

```
61335 LDY    667      Zeiger auf Ende des Empfangspuffers gleich
61338 INY
61339 CPY    668      dem Zeiger auf Anfang des Empfangspuffers (vgl. 61463)
61342 BEQ  61386      Ja: Ueberlauf des Empfangspuffers, weiter bei 61386
61344 STY    667      erhoehten Zeiger auf Ende des Empfangspuffers setzen
61347 DEY
61348 LDA    170      empfangenes Byte
61350 LDX    664      (Shift-Register wurde von rechts her gefuellt)
61353 CPX    #9
61355 BEQ  61361
61357 LSR             so oft nach rechts verschieben,
61358 INX             bis erstes empfangenes Bit an Bitposition 0 steht
61359 BNE  61353
61361 STA  (247),Y    empfangenes Byte im Empfangspuffer abspeichern
```

Verarbeitung des empfangenen Bytes (Fortsetzung)

```
61363 LDA    #32      Maske fuer Bit 5
61365 BIT    660      RS-232 Kommandoregister pruefen
61368 BEQ  61294      keine Uebertragung des Paritybits? Ja: weiter bei 61294
61370 BMI  61293      wird ein bestimmtes Bit statt Paritybit uebertragen? RTS
61372 LDA    167      empfangenes Paritybit
61374 EOR    171      mit errechnetem Wert verknuepfen
61376 BEQ  61381      Gleich? Ja: weiter bei 61381
61378 BVS  61293      Even Parity? Ja: alles in Ordnung, RTS
61380 BIT    ...
61381 BVC  61293      Odd Parity? Ja: alles in Ordnung, RTS
61383 LDA     #1      Bit 0, 'Parity Error'
61385 BIT    ...
61386 LDA     #4      Bit 2, 'Receiver Buffer Overrun'
61388 BIT    ...
61389 LDA   #128      Bit 7, 'Break Detected'
61391 BIT    ...
61392 LDA     #2      Bit 1, 'Framing Error'
61394 ORA    663      mit vorherigem Wert des RS-232 Statusregisters
61397 STA    663      verknuepfen und wieder abspeichern
61400 JMP  61310      > Empfang des neuen Bytes vorbereiten
```

Aufruf, falls Stoppbit = 0

```
61403 LDA    170      serielles Register fuer Byte-Empfang
61405 BNE  61392      wurden schon Bits empfangen? Ja: 'Framing Error'
61407 BEQ  61389      ansonsten 'Break Detected', weiter bei 61389
```

CHKOUT-Handling fuer RS-232

```
61409 STA    154      aktiven Ausgabekanal festsetzen
61411 LDA    660      RS-232 Kommandoregister
61414 LSR             Handshake-Mode
61415 BCC  61458      X-Line? Nein: weiter bei 61458
61417 LDA     #2      Maske fuer 'RTS OUT'
61419 BIT  56577      Port B (NMI-CIA) abfragen; liegt 'DSR IN' auf high?
61422 BPL  61453      Nein: 'DSR Signal Missing', RS-232 Status setzen
61424 BNE  61458      Liegt 'RTS OUT' (Sendeanfrage) auf high? Ja: fertig
61426 LDA    673      Werden momentan Daten empfangen?
61429 AND     #2
61431 BNE  61426      Ja: warten ...
61433 BIT  56577      Liegt 'CTS IN' auf low (Empfaenger bereit)?
61436 BVS  61433      Nein: warten ...
61438 LDA  56577
61441 ORA     #2
61443 STA  56577      'RTS OUT' auf high setzen (keine Sendeanfrage)
61446 BIT  56577      Liegt 'CTS IN' auf high?
61449 BVS  61458      Ja: fertig
61451 BMI  61446      zurueck nach 61446, solange 'DSR IN' auf high liegt
61453 LDA    #64
61455 STA    663      RS-232 Status auf 'DSR Signal Missing' setzen
61458 CLC
61459 RTS

61460 JSR  61480      > evtl. RS-232 Uebertragung in Gang bringen
```

CHROUT, Zeichen in RS-232 Sendepuffer bringen

```
61463 LDY    670      Zeiger auf naechste freie Stelle im RS-232 Sendepuffer
61466 INY             schon belegt (Test, ob Zeiger von unten den Ueber-
61467 CPY    669      tragungszeiger erreicht hat  Puffer wird "rundherum"
61470 BEQ  61460      benutzt)? Ja: warten  bis Platz im Puffer
```

SECOND, Ausgabe Sekundaeradresse nach LISTEN

```
60857 STA   149      Sekundaeradresse in Ausgaberegister
60859 JSR  60726     > Sekundaeradresse ausgeben
60862 LDA  56576     ATN := low (Kontroll-Modus beenden)
60865 AND   #247
60867 STA  56576
60870 RTS
```

TKSA, Ausgabe Sekundaeradresse nach TALK

```
60871 STA   149      Sekundaeradresse in Ausgaberegister
60873 JSR  60726     > Sekundaeradresse ausgeben
60876 SEI
60877 JSR  61088     > Serial Data := high
60880 JSR  60862     > ATN := low
60883 JSR  61061     > Clock Out := low (Kontroll-Modus beenden)
60886 JSR  61097     > Clock In = 0?
60889 BMI  60886     Nein: warten ...
60891 CLI
60892 RTS
```

CIOUT, Zeichen ausgeben (Pufferung fuer EOI)

```
60893 BIT   148      gepuffertes Zeichen vorhanden?
60895 BMI  60902     Ja: weiter bei 60902
60897 SEC
60898 ROR   148      Flag fuer 'Zeichen im Puffer' setzen
60900 BNE  60907     Unbedingter Sprung

60902 PHA
60903 JSR  60736     > Byte im Ausgaberegister auf Bus ausgeben
60906 PLA
60907 STA   149      Zeichen in Puffer bringen
60909 CLC
60910 RTS
```

UNTALK und UNLISTEN

```
60911 SEI
60912 JSR  61070     > Clock Out := high
60915 LDA  56576     ATN := high
60918 ORA     #8
60920 STA  56576
60923 LDA    #95     Kennzeichnung fuer UNTALK
60925 BIT    ...
60926 LDA    #63     Kennzeichnung fuer UNLISTEN
60928 JSR  60689     > Ausgabe mit ATN = high
60931 JSR  60862     > ATN := low (Kontroll-Modus beenden)
60934 TXA            XR merken
60935 LDX    #10     Verzoegerungsschleife um ca. 50 Mikrosekunden
60937 DEX
60938 BNE  60937
60940 TAX            XR wiederherstellen
60941 JSR  61061     > Clock Out := low
60944 JMP  61079     > Serial Data := low
```

CHRIN RS-232, Zeichen aus RS-232 Empfangspuffer holen

```
61574 LDA    663       RS-232 Statusbyte in Accu bringen
61577 LDY    668       Zeiger auf naechstes Zeichen im RS-232 Empfangspuffer
61580 CPY    667       gleich dem Zeiger auf das Ende des Empfangspuffers?
61583 BEQ  61596       Ja: keine Zeichen vorhanden, weiter bei 61596
61585 AND   #247       Bit 3 (Zeichen gueltig) im RS-232 Status loeschen
61587 STA    663       und in Register fuer RS-232 Status abspeichern
61590 LDA   (247),Y    Zeichen aus RS-232 Empfangspuffer lesen
61592 INC    668       Zeiger auf naechstes Zeichen im Empfangspuffer erhoehen
61595 RTS

61596 ORA     #8       Bit 3 (Empfangspuffer leer) im RS-232 Status setzen
61598 STA    663       und im Register fuer RS-232 Status abspeichern
61601 LDA     #0       Code null als Ersatzwert
61603 RTS
```

Warten, bis RS-232 Datenuebertragung abgeschlossen ist

```
61604 PHA              Accu merken
61605 LDA    673       Werden momentan Daten ueber den RS-232 Kanal uebertragen?
61608 BEQ  61627       Nein: fertig
61610 LDA    673       Abwarten, bis RS-232 Datenuebertragung abgeschlossen
61613 AND     #3
61615 BNE  61610
61617 LDA    #16       FLAG-NMI disablen
61619 STA  56589
61622 LDA     #0
61624 STA    673       RS-232 Register fuer aktive NMIs auf null setzen
61627 PLA              Accu wiederherstellen
61628 RTS
```

Meldungen zum I/O-Handling

```
61629  13  73  47  79  32  69  82  82  79  82  32 163       I/O ERROR #
61641  13  83  69  65  82  67  72  73  78  71 160           SEARCHING
61652  70  79  82 160                                       FOR
61656  13  80  82  69  83  83  32  80  76  65  89  32       PRESS PLAY
61668  79  78  32  84  65  80 197                           ON TAPE
61675  80  82  69  83  83  32  82  69  67  79  82  68  32   PRESS RECORD
61688  38  32  80  76  65  89  32                           & PLAY
61695  79  78  32  84  65  80 197                           ON TAPE
61702  13  76  79  65  68  73  78 199                       LOADING
61710  13  83  65  86  73  78  71 160                       SAVING
61718  13  86  69  82  73  70  89  73  78 199               VERIFYING
61728  13  70  79  85  78  68 160                           FOUND
61735  13  79  75 141                                       OK
```

Ausgabe von I/O-Meldungen

```
61739 BIT    157      Ausgabe unterdruecken (Direktmodus)?
61741 BPL  61756      Ja: Ruecksprung, RTS
61743 LDA  61629,Y    Meldung im Bereich von
61746 PHP             von 61629 bis 61737 ausgeben
61747 AND   #127
61749 JSR  65490      > CHROUT, Ausgabe auf aktiven Ausgabe-Kanal
61752 INY
61753 PLP             Bit 7 im Zeichen gesetzt?
61754 BPL  61743      Nein: weiter ausgeben
61756 CLC
61757 RTS
```

GETIN, Zeichen vom aktiven Eingabe-Kanal in Accu

```
61758 LDA    153      aktiver Eingabe-Kanal
61760 BNE  61770      =0 (Tastatur)? Nein: weiter bei 61770
61762 LDA    198      Anzahl Zeichen im Tastaturpuffer
61764 BEQ  61781      =0? Ja: RTS
61766 SEI             Interrupt verhindern
61767 JMP  58804      Tastaturpuffer um ein Zeichen abbauen

61770 CMP     #2      aktiver Eingabe-Kanal = 2 (RS-232 Kanal)?
61772 BNE  61798      Nein: weiter bei 61798
61774 STY    151      YR merken
61776 JSR  61574      Zeichen aus RS-232 Empfangspuffer holen
61779 LDY    151      YR wiederherstellen
61781 CLC
61782 RTS
```

CHRIN, Zeichen vom aktiven Eingabe-Kanal in Accu

```
61783 LDA    153      Eingabe von Tastatur?
61785 BNE  61798      Nein: weiter bei 61798
61787 LDA    211      Cursorspaltenpointer bei Aufruf
61789 STA    202      nach (202) und
61791 LDA    214      Cursorzeilenpointer bei Aufruf
61793 STA    201      nach (201) speichern
61795 JMP  58930      > Eingabeschleife zur Zeichenuebernahme vom Bildschirm

61798 CMP     #3      aktiver Eingabe-Kanal = 3 (Bildschirm)?
61800 BNE  61811      Nein: weiter bei 61811
61802 STA    208      Flag fuer 'Zeichen direkt vom Bildschirm lesen'
61804 LDA    213      aktuelle Cursorzeilenlaenge
61806 STA    200      als Pointer fuer Ende der Zeile speichern
61808 JMP  58930      > Zeichenuebernahme vom Bildschirm
```

CHRIN, Zeichen vom aktiven Eingabe-Kanal in Accu (Fortsetzung)

```
61811 BCS  61869       Eingabe-Kanal > 3 (serieller Bus)? Ja: weiter bei 61869
61813 CMP     #2       gleich 2 (RS-232 Kanal)?
61815 BEQ  61880       Ja: weiter bei 61880
61817 STX    151       XR merken
61819 JSR  61849       > Zeichen aus Puffer lesen, evtl. Block in Puffer lesen
61822 BCS  61846       Fehler? Ja: weiter bei 61846
61824 PHA              gelesenes Zeichen merken
61825 JSR  61849       > Zeichen aus Puffer lesen, evtl. Block in Puffer lesen
61828 BCS  61843       Fehler? Ja: weiter bei 61843
61830 BNE  61837       Code = 0? Nein: weiter bei 61837
61832 LDA    #64       Bit 6 fuer 'End or Indentify'
61834 JSR  65052       > I/O-Status setzen
61837 DEC    166       Pufferzeiger vermindern
61839 LDX    151       XR wiederherstellen
61841 PLA              gelesenes Zeichen wiederherstellen
61842 RTS

61843 TAX              Zeichencode im Accu merken
61844 PLA              Stack normalisieren
61845 TXA              und den Zeichencode wieder in den Accu bringen
61846 LDX    151       XR wiederherstellen
61848 RTS

61849 JSR  63501       > Pufferzeiger erhoehen; Test, ob Pufferende erreicht
61852 BNE  61865       Puffer leer? Nein: weiter bei 61865
61854 JSR  63553       > Block von Band in Puffer lesen
61857 BCS  61876       Fehler? Ja: RTS mit gesetztem Carry
61859 LDA     #0
61861 STA    166       Pufferzeiger auf erstes Zeichen setzen
61863 BEQ  61849       Unbedingter Sprung

61865 LDA   (178),Y    Zeichen aus Puffer lesen
61867 CLC              Carry fuer 'kein Fehler' loeschen
61868 RTS

61869 LDA    144       I/O-Statusbyte = 0?
61871 BEQ  61877       Ja: weiter bei 61877
61873 LDA    #13       Code fuer "RETURN"
61875 CLC
61876 RTS

61877 JMP  60947       > ACPTR, Zeichen vom seriellen Bus holen

61880 JSR  61774       > Zeichen aus RS-232 Empfangspuffer holen
61883 BCS  61876       Fehler? Ja: RTS mit gesetztem Carry
61885 CMP     #0       Code = 0?
61887 BNE  61875       Nein: Carry loeschen, RTS
61889 LDA    663       RS-232 Status lesen
61892 AND    #96       Bit 5 oder Bit 6 gesetzt?
61894 BNE  61873       Ja: weiter bei 61873
61896 BEQ  61880       ansonsten zurueck nach 61880
```

CHROUT, Ausgabe auf aktiven Kanal

```
61898 PHA              Datenbyte retten
61899 LDA    154       aktiver Ausgabe-Kanal
61901 CMP     #3       =3 (Bildschirm)?
61903 BNE  61909       Nein: weiter bei 61909
61905 PLA              Datenbyte wieder holen
61906 JMP  59158       > Ausgabe auf Bildschirm
```

CHROUT, Ausgabe auf aktiven Kanal (Fortsetzung)

```
61909 BCC  61915        Ausgabe auf seriellen Bus? Nein: weiter bei 61915
61911 PLA               Datenbyte wiederherstellen
61912 JMP  60893        > CIOUT, Ausgabe des Accus auf seriellen Bus

61915 LSR               Bit 0 der Ausgabe-Kanal-Nummer in die Carry-Flag schieben
61916 PLA               Datenbyte wiederherstellen
61917 STA   158         und zwischenspeichern
61919 TXA               Indexregister auf Stack retten
61920 PHA
61921 TYA
61922 PHA
61923 BCC  61960        Ausgabe auf Band? Nein: weiter bei 61960
61925 JSR  63501        > Pufferzeiger erhoehen; Test, ob Pufferende erreicht
61928 BNE  61944        Puffer voll? Nein: weiter bei 61944
61930 JSR  63588        > Block von Puffer auf Band schreiben
61933 BCS  61949        Fehler? Ja: weiter bei 61949
61935 LDA    #2         Kennzeichnung fuer Datenblock
61937 LDY    #0
61939 STA  (178),Y      an erste Cassettenpufferposition schreiben
61941 INY
61942 STY   166         Cassettenpufferzeiger initialisieren
61944 LDA   158         Datenbyte bei Aufruf von CHROUT
61946 STA  (178),Y      im Puffer abspeichern
61948 CLC               Carry fuer 'kein Fehler' loeschen
61949 PLA               Indexregister wiederherstellen
61950 TAY
61951 PLA
61952 TAX
61953 LDA   158         Datenbyte wiederherstellen
61955 BCC  61959        Fehler aufgetreten? Nein: RTS
61957 LDA    #0         Accu := 0, Fehlercode fuer "BREAK", Abbruch durch RUNSTOP
61959 RTS

61960 JSR  61463        > Zeichen in RS-232 Sendepuffer schreiben
61963 JMP  61948        > Abschluss CHROUT
```

CHKIN, Vorbereitungen fuer Datenempfang

```
61966 JSR  62223        > Fileparameter in Tabelle suchen
61969 BEQ  61974        gefunden? Ja: weiter bei 61974
61971 JMP  63233        > I/O-Error #3, "FILE NOT OPEN"

61974 JSR  62239        > Fileparameter aktualisieren
61977 LDA   186         Geraetenummer
61979 BEQ  62003        = 0 (Tastatur)? Ja: weiter bei 62003
61981 CMP    #3         = 3 (Bildschirm)?
61983 BEQ  62003        Ja: weiter bei 62003
61985 BCS  62007        Geraetenummer > 3 (serieller Bus)? Ja: weiter bei 62007
61987 CMP    #2         = 2 (RS-232 Kanal)?
61989 BNE  61994        Nein: weiter bei 61994
61991 JMP  61517        > Eingabe vom RS-232 Kanal vorbereiten

61994 LDX   185         aktuelle Sekundaeradresse
61996 CPX   #96         = 96 (einzige Sekundaeradresse beim Lesen von Tape)?
61998 BEQ  62003        Ja: weiter bei 62003
62000 JMP  63242        > I/O-Error #6, "NOT OUTPUT FILE"

62003 STA   153         Geraetenummer als aktiven Eingabe-Kanal speichern
62005 CLC
62006 RTS
```

CHKIN, Vorbereitungen fuer Datenempfang (Fortsetzung)

```
62007 TAX               Geraetenummer ins XR retten
62008 JSR  60681        > TALK
62011 LDA    185        aktuelle Sekundaeradresse
62013 BPL  62021        kleiner 128? Ja: weiter bei 62021
62015 JSR  60876        > Kontrollmodus beenden
62018 JMP  62024        > naechsten Befehl ueberspringen

62021 JSR  60871        > TKSA, Sekundaeradresse auf Bus ausgeben
62024 TXA               Geraetenummer auf Bus ausgeben
62025 BIT    144        Bit 7 im I/O-Statusbyte gesetzt?
62027 BPL  62003        Nein: Abschluss CHKIN
62029 JMP  63239        > I/O-Error #5, "DEVICE NOT PRESENT ERROR"
```

CHKOUT, Ausgabevorbereitungen

```
62032 JSR  62223        > Filenummer in Tabelle suchen
62035 BEQ  62040        gefunden? Ja: weiter bei 62040
62037 JMP  63233        > I/O-Error #3, "FILE NOT FOUND"

62040 JSR  62239        > Fileparameter aktualisieren
62043 LDA    186        Geraetenummer
62045 BNE  62050        = 0 (Tastatur)? Nein: weiter bei 62050
62047 JMP  63245        > I/O-Error #7, "NOT OUTPUT FILE"

62050 CMP     #3        Geraetenummer
62052 BEQ  62069        = 3 (Bildschirm)? Ja: weiter bei 62069
62054 BCS  62073        groesser 3 (serieller Bus)? Ja: weiter bei 62073
62056 CMP     #2        = 2 (RS-232 Kanal)?
62058 BNE  62063        Nein: weiter bei 62063
62060 JMP  61409        > Ausgabe auf RS-232 Kanal vorbereiten

62063 LDX    185        aktuelle Sekundaeradresse
62065 CPX    #96        = 96 (einzige Sekundaeradresse zum Lesen von Tape)?
62067 BEQ  62047        Ja: I/O-Error #7, "NOT OUTPUT FILE"
62069 STA    154        Geraetenummer als aktiven Ausgabe-Kanal speichern
62071 CLC
62072 RTS

62073 TAX               Geraetenummer ins XR retten
62074 JSR  60684        > LISTEN
62077 LDA    185        aktuelle Sekundaeradresse
62079 BPL  62086        kleiner 128? Ja: weiter bei 62086
62081 JSR  60862        > Kontrollmodus beenden
62084 BNE  62089        Unbedingter Sprung

62086 JSR  60857        > SECOND, Sekundaeradresse auf Bus ausgeben
62089 TXA               Geraetenummer wiederherstellen
62090 BIT    144        Bit 7 im I/O-Statusbyte gesetzt?
62092 BPL  62069        Nein: weiter bei 62069
62094 JMP  63239        > I/O-Error #5, "DEVICE NOT PRESENT"
```

CLOSE, File (Nummer im Accu) schliessen

```
62097 JSR  62228        > Filenummer in Tabelle suchen
62100 BEQ  62104        gefunden? Ja: weiter bei 62104
62102 CLC
62103 RTS
```

CLOSE, File (Nummer im Accu) schliessen (Fortsetzung)

```
62104 JSR  62239       > Fileparameter aktualisieren
62107 TXA              Pointer auf Parametereintrag in Filetabelle
62108 PHA              auf Stack legen
62109 LDA   186        Geraetenummer
62111 BEQ  62193       = 0 (Tastatur)? Ja: weiter bei 62193
62113 CMP    #3        = 3 (Bildschirm)?
62115 BEQ  62193       Ja: weiter bei 62193
62117 BCS  62190       Geraetenummer > 3 (serieller Bus)? Ja: weiter bei 62190
62119 CMP    #2        = 2 (RS-232 Kanal)?
62121 BNE  62152       Nein: weiter bei 62152
62123 PLA              Pointer auf Eintrag in Filetabelle vom Stack holen
62124 JSR  62194       > File-Eintrag aus Tabelle loeschen
62127 JSR  62595       > RS-232 Uebertragung beenden
62130 JSR  65063       > MEMTOP, Lesen des Speicherendes
62133 LDA   248        Highbyte des Pointers auf den RS-232 Empfangspuffer
62135 BEQ  62138       = 0? Ja: weiter bei 62138
62137 INY              Pointer high auf Ende Arbeitsspeicher um eins erhoehen
62138 LDA   250        Highbyte des Pointers auf den RS-232 Sendepuffer
62140 BEQ  62143       = 0? Ja: weiter bei 62143
62142 INY              Pointer high auf Ende Arbeitsspeicher um eins erhoehen
62143 LDA    #0
62145 STA   248
62147 STA   250        Highbytes der RS-232 Puffer-Pointer auf Null setzen
62149 JMP  62589       > weiter bei 62589

62152 LDA   185        aktuelle Sekundaeradresse
62154 AND   #15        Bit 0 bis Bis 3 isolieren
62156 BEQ  62193       Lesen von Cassette? Ja: weiter bei 62193
62158 JSR  63440       > Pointer auf Cassettenpuffer nach (XR/YR) bringen
62161 LDA    #0        Markierung fuer 'letztes Byte im Datenfile'
62163 SEC              Flag fuer Ausgabe auf Recorder
62164 JSR  61917       > Zeichen in Cassettenpuffer bringen
62167 JSR  63588       > letzten Block auf Band schreiben
62170 BCC  62176       Fehler? Nein: weiter bei 62176
62172 PLA              Pointer auf Eintrag in Filetabelle vom Stack holen
62173 LDA    #0        Accu := 0, Fehlercode fuer "BREAK", Abbruch durch RUNSTOP
62175 RTS

62176 LDA   185        aktuelle Sekundaeradresse
62178 CMP   #98        = 98 (OPEN mit EOT-Kennzeichnung)?
62180 BNE  62193       Nein: weiter bei 62193
62182 LDA    #5        Kennzeichnung fuer EOT-Block
62184 JSR  63338       > EOT-Block auf Band schreiben
62187 JMP  62193       > weiter bei 62193

62190 JSR  63042       > CLOSE auf seriellen Bus ausgeben
62193 PLA              Pointer auf Eintrag in Filetabelle vom Stack holen
62194 TAX              als Index ins XR bringen
62195 DEC   152        Zaehler fuer Anzahl an offenen Files um eins vermindern
62197 CPX   152        und mit Pointer auf Eintrag in Filetabelle vergleichen
62199 BEQ  62221       Gleich? Ja: fertig
62201 LDY   152        Pointer auf letzten Eintrag in Filetabelle
62203 LDA   601,Y      letzten Tabelleneintrag in freiwerdende Stelle
62206 STA   601,X      (Eintrag der zu loeschenden Datei) uebertragen
62209 LDA   611,Y
62212 STA   611,X
62215 LDA   621,Y
62218 STA   621,X
62221 CLC
62222 RTS
```

Filenummer in Filetabelle suchen

```
62223 LDA     #0
62225 STA    144       Status loeschen
62227 TXA              gesuchte Filenummer von XR in Accu bringen
62228 LDX    152       Zeiger in File-Tabelle
62230 DEX              bereits alle Eintraege durchsucht?
62231 BMI   62254      Ja: RTS
62233 CMP    601,X     Accu mit Filenummer in Tabelle vergleichen
62236 BNE   62230      Ungleich? Ja: weitersuchen
62238 RTS
```

Fileparameter aktualisieren

```
62239 LDA    601,X     Filenummer aus Tabelle
62242 STA    184       als aktuelle Filenummer speichern
62244 LDA    611,X
62247 STA    186       ebenso die Primaeradresse
62249 LDA    621,X
62252 STA    185       und die Sekundaeradresse
62254 RTS
```

CLALL, alle Files loeschen

```
62255 LDA     #0
62257 STA    152       alle Files loeschen
```

CLRCHN, aktive I/O-Kanaele schliessen, Standardkanaele aktivieren

```
62259 LDX     #3       aktiver Ausgabekanal
62261 CPX    154       groesser als 3 (serieller Bus)?
62263 BCS   62268      Nein: weiter bei 62268
62265 JSR   60926      > UNLISTEN auf Bus ausgeben
62268 CPX    153       Eingabekanal groesser als 3?
62270 BCS   62275      Nein: weiter bei 62275
62272 JSR   60911      > UNTALK auf Bus ausgeben
62275 STX    154       aktiver Ausgabe-Kanal := 3 (Bildschirm)
62277 LDA     #0
62279 STA    153       aktiver Eingabe-Kanal := 0 (Tastatur)
62281 RTS
```

OPEN, logisches File eroeffnen

```
62282 LDX    184       Filenummer
62284 BNE   62289      = 0 (nicht erlaubt!)? Nein: weiter bei 62289
62286 JMP   63242      > I/O-Error #6, "NOT INPUT FILE"

62289 JSR   62223      > Filenummer in Tabelle suchen
62292 BNE   62297      gefunden? Nein: weiter bei 62297
62294 JMP   63230      > I/O-Error #2, "FILE OPEN"

62297 LDX    152       Anzahl offene Files
62299 CPX    #10       = 10 (maximale Anzahl)?
62301 BCC   62306      Nein: weiter bei 62306
62303 JMP   63227      > I/O-Error #1, "TOO MANY FILES"
```

OPEN, logisches File eroeffnen (Fortsetzung)

```
62306 INC    152      Anzahl offene Files um eins erhoehen
62308 LDA    184      Filenummer hinter letzten Tabelleneintrag
62310 STA    601,X    in Tabelle der Fileparameter schreiben
62313 LDA    185
62315 ORA    #96      Bit 5 und 6 der Sekundaeradresse setzen,
62317 STA    185      als neue Sekundaeradresse speichern
62319 STA    621,X    und in Tabelle bringen
62322 LDA    186      ebenso die Geraetenummer
62324 STA    611,X    in die Tabelle bringen
62327 BEQ   62419     Geraetenummer = 0 (Tastatur)? Ja: fertig
62329 CMP     #3      = 3 (Bildschirm)?
62331 BEQ   62419     Ja: weiter bei 62419
62333 BCC   62340     kleiner 3? Ja: weiter bei 62340
62335 JSR   62421     > OPEN auf seriellen Bus ausgeben
62338 BCC   62419     Unbedingter Sprung zum Abschluss von OPEN

62340 CMP     #2      Geraetenummer = 2 (RS-232 Kanal)?
62342 BNE   62347     Nein: weiter bei 62347
62344 JMP   62473     > OPEN-Handling fuer RS-232

62347 JSR   63440     > Pointer auf Cassettenpuffers nach (XR/YR)
62350 BCS   62355     Ist (XR/YR) kleiner 512? Nein: weiter bei 62355
62352 JMP   63251     > I/O-Error #9, "ILLEGAL DEVICE NUMBER"

62355 LDA    185      aktuelle Sekundaeradresse
62357 AND    #15      Bit 0 bis Bit 3 isolieren
62359 BNE   62392     Lesen von Tape? Nein: weiter bei 62392
62361 JSR   63511     > "PRESS PLAY ON TAPE"
62364 BCS   62420     Abbruch durch RUNSTOP? Ja: RTS
62366 JSR   62895     > Im Direktmodus "SEARCHING" und Filenamen ausgeben
62369 LDA    183      Laenge des Filenamens = 0?
62371 BEQ   62383     Ja: weiter bei 62383
62373 JSR   63466     > Dateikopf mit verlangtem Namen suchen
62376 BCC   62402     Fehler? Nein: weiter bei 62402
62378 BEQ   62420     Abbruch durch RUNSTOP? Ja: RTS
62380 JMP   63236     > I/O-Error #4, "FILE NOT FOUND"

62383 JSR   63276     > Datenblock in Puffer lesen
62386 BEQ   62420     'End of Tape' oder RUNSTOP? Ja: RTS
62388 BCC   62402     Fehler? Nein: weiter bei 62402
62390 BCS   62380     I/O-Error #4, "FILE NOT FOUND"
                      (Die Branches an den Adressen 62386 und 62388 sollten
                      wohl eigentlich in umgekehrter Reihenfolge stehen!)

62392 JSR   63544     > "PRESS RECORD & PLAY ON TAPE"
62395 BCS   62420     Abbruch durch RUNSTOP? Ja: RTS
62397 LDA     #4      Kennzeichnung fuer Dateikopf einer sequentiellen Datei
62399 JSR   63338     > Dateikopf in Tapebuffer erzeugen und auf Band schreiben
62402 LDA   #191      Kennzeichnung fuer 'kein Zeichen im Puffer'
62404 LDY    185      aktuelle Sekundaeradresse
62406 CPY    #96      = 96 (Lesen von Cassette)?
62408 BEQ   62417     Ja: weiter bei 62417
62410 LDY     #0      Index fuer erste Position des Cassettenpuffers
62412 LDA     #2      Kennzeichnung fuer Datenblock einer sequentiellen Datei
62414 STA   (178),Y   in erste Pufferposition bringen
62416 TYA
62417 STA    166      Pufferzeiger festsetzen
62419 CLC
62420 RTS
```

OPEN auf seriellen Bus ausgeben

```
62421 LDA    185       aktuelle Sekundaeradresse
62423 BMI   62419      kleiner 128? Nein: fertig
62425 LDY    183       Laenge des Filenamens
62427 BEQ   62419      = 0? Ja: fertig
62429 LDA     #0
62431 STA    144       Status loeschen
62433 LDA    186       Geraetenummer
62435 JSR   60684      > LISTEN
62438 LDA    185       aktuelle Sekundaeradresse
62440 ORA   #240       Bit 4 bis Bit 7 setzen (Kennzeichnung fuer OPEN)
62442 JSR   60857      > SECOND, Ausgabe Sekundaeradresse
62445 LDA    144       Bit 7 im I/O-Statusbyte gesetzt?
62447 BPL   62454      Nein: weiter bei 62454
62449 PLA              Ruecksprungadresse vom Stack entfernen
62450 PLA
62451 JMP   63239      > I/O-Error #5, "DEVICE NOT PRESENT"

62454 LDA    183       Laenge des Filenamens
62456 BEQ   62470      = 0? Ja: weiter bei 62470
62458 LDY     #0       Index fuer Filenamen
62460 LDA   (187),Y    Zeichen aus Filenamen lesen
62462 JSR   60893      > CIOUT, Zeichen auf seriellen Bus ausgeben
62465 INY
62466 CPY    183       alle Zeichen ausgegeben?
62468 BNE   62460      Nein: weitermachen ...
62470 JMP   63060      > UNLISTEN, Carry loeschen, RTS
```

OPEN fuer RS-232-Handling

```
62473 JSR   62595      > saemtliche NMIs verhindern, RS-232 I/O initialisieren
62476 STY    663       RS-232 Status loeschen
62479 CPY    183       Laenge des Filenamens
62481 BEQ   62493      = 0? Ja: weiter bei 62493
62483 LDA   (187),Y    Uebertragung der ersten 4 Bytes des Filenamens
62485 STA    659,Y     in den Bereich (659,...,662)
62488 INY
62489 CPY     #4
62491 BNE   62479
62493 JSR   61258      > Wortlaenge fuer RS-232 Datenuebertragung feststellen
62496 STX    664       und in den Speicher fuer die Wortlaenge schreiben
62499 LDA    659       RS-232 Kontrollregister
62502 AND    #15       Bits 0 bis 3 isolieren (Kennzeichnung fuer die Baud-Rate)
62504 BEQ   62534      = 0 (User-Baud-Rate)? Ja: weiter bei 62534
62506 ASL              Index fuer Tabelle verdoppeln
62507 TAX              und als Index ins XR bringen
62508 LDA    678       Flag fuer Quarzfrequenz (NTSC=0, PAL=1)
62511 BNE   62522      = 1? Ja: weiter bei 62522
62513 LDY   65217,X    Timer-Werte fuer die Baud-Rate aus
62516 LDA   65216,X    NTSC-Tabelle holen
62519 JMP   62528      > weiter bei 62528

62522 LDY   58603,X    Timer-Werte fuer die Baud-Rate aus
62525 LDA   58602,X    PAL-Tabelle holen
62528 STY    662       und nach (661/662) bringen
62531 STA    661
62534 LDA    661
62537 ASL              Timer-Wert in (661/662) verdoppeln,
62538 JSR   65326      > 200 addieren und nach (665/666) uebertragen
62541 LDA    660       RS-232 Kommandoregister
62544 LSR              Handshake-Mode
62545 BCC   62556      X-Line? Nein: weiter bei 62556
```

OPEN fuer RS-232-Handling (Fortsetzung)

```
62547 LDA  56577       Port B der NMI-CIA
62550 ASL              Ist 'DSR IN' auf high gesetzt?
62551 BCS  62556       Ja: weiter bei 62556
62553 JSR  61453       > 'DSR Signal Missing', RS-232 Status setzen
62556 LDA    667       Pointer auf Ende des Empfangspuffers
62559 STA    668       in Pointer auf Anfang des Empfangspuffers uebertragen
62562 LDA    670       Pointer auf Ende des Sendepuffers
62565 STA    669       in Pointer auf Anfang des Sendepuffers uebertragen
62568 JSR  65063       > MEMTOP, Speicherende lesen (XR/YR)
62571 LDA    248       Highbyte des Pointers auf den Empfangspuffer
62573 BNE  62580       = 0? Nein: weiter bei 62580
62575 DEY
62576 STY    248       256 Bytes fuer den Empfangspuffer reservieren
62578 STX    247
62580 LDA    250       Highbyte des Pointers auf den Sendepuffer
62582 BNE  62589       = 0? Nein: weiter bei 62589
62584 DEY
62585 STY    250       256 Bytes fuer den Sendepuffer reservieren
62587 STX    249

62589 SEC              Fehlerflag setzen
62590 LDA   #240       Code fuer 'Puffer schuetzen/freigeben', vgl. 57593 ff
62592 JMP  65069       > MEMTOP, Speicherende festsetzen (XR/YR)
```

RS-232-I/O initialisieren/abbrechen

```
62595 LDA   #127
62597 STA  56589       saemtliche NMIs (ausser RESTORE) blockieren
62600 LDA     #6
62602 STA  56579       Datenrichtungsregister
62605 STA  56577       und Port fuer RS-232-Handling initialisieren
62608 LDA     #4
62610 ORA  56576
62613 STA  56576       'RS-232 OUT' auf high setzen
62616 LDY     #0
62618 STY    673       RS-232-Register fuer aktive Interrupts auf null setzen
62621 RTS
```

LOAD, Load und Verify von Programmen

```
62622 STX    195       Anfangsadresse des Programms
62624 STY    196       falls APPEND (z. B. an Speicheranfang)
62626 JMP  (816)       Normalwert des Vektors (816/817): 62629
62629 STA    147       Flag fuer LOAD (0) und VERIFY (1)
62631 LDA     #0
62633 STA    144       Status loeschen
62635 LDA    186       Geraetenummer
62637 BNE  62642       =0 (Tastatur)? Nein: weiter bei 62642
62639 JMP  63251       > I/O-Error #9, "ILLEGAL DEVICE NUMBER"

62642 CMP     #3       =3 (Bildschirm)?
62644 BEQ  62639       Ja: I/O-Error #9, "ILLEGAL DEVICE NUMBER"
62646 BCC  62771       kleiner 3? Ja: weiter bei 62771
62648 LDY    183       Laenge des Filenamens
62650 BNE  62655       = 0? Nein: weiter bei 62655
62652 JMP  63248       > I/O-Error #8, "MISSING FILE NAME"
```

LOAD, Load und Verify von Programmen (Fortsetzung)

```
62655 LDX    185        Sekundaeradresse ins XR retten
62657 JSR   62895       > Ausgabe "SEARCHING ", evtl. "FOR" mit Filenamen
62660 LDA    #96
62662 STA    185        Sekundaeradresse auf 96 (LOAD vom seriellen Bus) setzen
62664 JSR   62421       > OPEN auf seriellen Bus ausgeben
62667 LDA    186        Geraetenummer in Accu
62669 JSR   60681       > TALK
62672 LDA    185        Sekundaeradresse in Accu
62674 JSR   60871       > TKSA, Sekundaeradresse auf Bus ausgeben
62677 JSR   60947       > ACPTR, Zeichen von Bus holen
62680 STA    174        und in den Ladezeiger low bringen
62682 LDA    144        I/O-Statusbyte
62684 LSR
62685 LSR               Zeitfehler beim Lesen?
62686 BCS   62768       Ja: I/O-Error #4, "FILE NOT FOUND"
62688 JSR   60947       > ACPTR, Zeichen von Bus holen
62691 STA    175        und in den Ladezeiger high bringen
62693 TXA               Sekundaeradresse bei Aufruf von 'LOAD'
62694 BNE   62704       ungleich null (Absolut-LOAD)? Ja: weiter bei 62704
62696 LDA    195        ansonsten APPEND-Adresse bei Aufruf von 'LOAD'
62698 STA    174        in den Ladezeiger uebertragen
62700 LDA    196
62702 STA    175
62704 JSR   62930       > Ausgabe "LOADING"/"VERIFYING"
62707 LDA   #253        Bit 1 (Zeitfehler beim Lesen)
62709 AND    144
62711 STA    144        im Status loeschen
62713 JSR   65505       > STOP, Abfrage der RUNSTOP-Taste
62716 BNE   62721       RUNSTOP gedrueckt? Nein: weiter bei 62721
62718 JMP   63027       > CLOSE auf Bus ausgeben, Flag fuer Abbruch setzen

62721 JSR   60947       > ACPTR, Zeichen vom Bus holen
62724 TAX               und ins XR retten
62725 LDA    144        I/O-Statusbyte
62727 LSR
62728 LSR               Bit 1 (Zeitfehler beim Lesen) gesetzt?
62729 BCS   62707       Ja: nochmal probieren
62731 TXA               gelesenes Zeichen wieder in Accu bringen
62732 LDY    147        Flag fuer 'VERIFY' gesetzt?
62734 BEQ   62748       Nein: weiter bei 62748
62736 LDY    #0         Index fuer Pointer
62738 CMP   (174),Y     gelesenes Byte mit Byte im Arbeitsspeicher vergleichen
62740 BEQ   62750       Gleich? Ja: weiter bei 62750
62742 LDA    #16        Bit 4 fuer Nichtuebereinstimmung
62744 JSR   65052       > Status setzen
62747 BIT    ...
62748 STA   (174),Y     gelesenes Byte im Arbeitsspeicher ablegen
62750 INC    174        Lesepointer erhoehen
62752 BNE   62756
62754 INC    175
62756 BIT    144        I/O-Statusbyte
62758 BVC   62707       Bit 6 (End or Indentify) gesetzt? Nein: weiterlesen ...
62760 JSR   60911       > UNTALK
62763 JSR   63042       > CLOSE auf seriellen Bus ausgeben
62766 BCC   62889       Fehler? Nein: weiter bei 62889
62768 JMP   63236       > I/O-Error #4, "FILE NOT FOUND"

62771 LSR               Bit 0 der Geraetenummer in die Carry-Flag schieben
62772 BCS   62777       'LOAD' von Recorder? Ja: weiter bei 62777
62774 JMP   63251       > I/O-Error #9, "ILLEGAL DEVICE NUMBER"
```

LOAD, Load und Verify von Programmen (Fortsetzung)

```
62777 JSR  63440       > Pointer auf Cassettenpuffer nach (XR/YR)
62780 BCS  62785       Ist (XR/YR) kleiner 512? Nein: weiter bei 62785
62782 JMP  63251       > I/O-Error #9, "ILLEGAL DEVICE NUMBER"

62785 JSR  63511       > "PRESS PLAY ON TAPE"
62788 BCS  62894       Abbruch durch RUNSTOP? Ja: RTS
62790 JSR  62895       > Im Direktmodus "SEARCHING " und Filenamen ausgeben
62793 LDA   183        Laenge des Filenamens
62795 BEQ  62806       = 0? Ja: weiter bei 62806
62797 JSR  63466       > Dateikopf mit verlangtem Namen suchen
62800 BCC  62813       Fehler? Nein: weiter bei 62813
62802 BEQ  62894       Abbruch durch RUNSTOP? Ja: RTS
62804 BCS  62768       I/O-Error #4, "FILE NOT FOUND"

62806 JSR  63276       > Dateikopf in Puffer lesen
62809 BEQ  62894       RUNSTOP oder 'End of Tape'? Ja: RTS (vgl. 62383 ff)
62811 BCS  62768       I/O-Error #4, "FILE NOT FOUND"

62813 LDA   144        I/O-Statusbyte
62815 AND   #16        Bit 4 (Lesefehler) isolieren
62817 SEC              Flag fuer Fehler setzen
62818 BNE  62894       Bit 4 gesetzt? Ja: RTS
62820 CPX    #1        Relativprogramm (meist BASIC)?
62822 BEQ  62841       Ja: weiter bei 62841
62824 CPX    #3        Absolutprogramm (kein APPEND an Speicheranfang!)?
62826 BNE  62793       Nein: weitersuchen ...
62828 LDY    #1
62830 LDA  (178),Y     Startadresse aus Cassettenpuffer
62832 STA   195        in Ladezeiger bringen
62834 INY
62835 LDA  (178),Y
62837 STA   196
62839 BCS  62845       Unbedingter Sprung

62841 LDA   185        Sekundaeradresse = 0 (APPEND)?
62843 BNE  62828       Nein: weiter bei 62828
62845 LDY    #3
62847 LDA  (178),Y     Endadresse+1 low des Programms
62849 LDY    #1
62851 SBC  (178),Y     minus Startadresse low des Programms
62853 TAX
62854 LDY    #4
62856 LDA  (178),Y     ebenso high
62858 LDY    #2
62860 SBC  (178),Y     ergibt Laenge des Programms
62862 TAY
62863 CLC
62864 TXA              Programmlaenge zur
62865 ADC   195        APPEND-Adresse bei Aufruf von 'LOAD' addieren,
62867 STA   174        ergibt die eigentliche Startadresse bei LOAD oder VERIFY
62869 TYA
62870 ADC   196
62872 STA   175
62874 LDA   195        (195/196) nach (193/194) uebertragen (Ladezeiger)
62876 STA   193
62878 LDA   196
62880 STA   194
62882 JSR  62930       > Ausgabe "LOADING" bzw. "VERIFYING"
62885 JSR  63562       > Programm einlesen
62888 BIT   ...
62889 CLC
```

LOAD, Load und Verify von Programmen (Fortsetzung)

```
62890 LDX    174       Endadresse+1 des gelesenen Programms
62892 LDY    175       nach (XR/YR) uebertragen
62894 RTS
```

Im Direktmodus Ausgabe "SEARCHING " und evtl. "FOR " mit Filenamen

```
62895 LDA    157       Ausgabe unterdruecken (Direktmodus)?
62897 BPL  62929       Ja: RTS
62899 LDY    #12       Offset fuer "SEARCHING "
62901 JSR  61743       > Meldung drucken
62904 LDA    183       Laenge des Filenamens
62906 BEQ  62929       = 0? Ja: RTS
62908 LDY    #23       Offset fuer "FOR "
62910 JSR  61743       > Meldung drucken
62913 LDY    183       Laenge des Filenamens
62915 BEQ  62929       = 0? Ja: RTS
62917 LDY     #0       Index fuer Filenamen auf null setzen
62919 LDA   (187),Y    Zeichen aus Filenamen lesen
62921 JSR  65490       > CHROUT, Zeichen im Accu ausgeben
62924 INY
62925 CPY    183       alle Zeichen ausgegeben?
62927 BNE  62919       Nein: weitermachen ...
62929 RTS
```

Ausgabe von "LOADING" bzw. "VERIFYING"

```
62930 LDY    #73       Offset fuer "LOADING"
62932 LDA    147       Flag fuer 'VERIFY' gesetzt?
62934 BEQ  62938       Nein: weiter bei 62938
62936 LDY    #89       Offset fuer "VERIFYING"
62938 JMP  61739       > Meldung ausgeben
```

SAVE, Speichern von Programmen

```
62941 STX    174       Endadresse des Programms nach (174/175)
62943 STY    175
62945 TAX              Wert im Accu entspricht Offset fuer Endpointer
62946 LDA      0,X     Startadresse low aus Pointer
62948 STA    193       nach (193)
62950 LDA      1,X     und Startadresse high
62952 STA    194       nach (194) uebertragen
62954 JMP   (818)      Normalwert des Vektors (818/819): 62957
62957 LDA    186       Geraetenummer
62959 BNE  62964       = 0 (Tastatur)? Nein: weiter bei 62964
62961 JMP  63251       > I/O-Error #9, "ILLEGAL DEVICE NUMBER"

62964 CMP     #3       Geraetenummer = 3 (Bildschirm)?
62966 BEQ  62961       Ja: I/O-Error #9, "ILLEGAL DEVICE NUMBER"
62968 BCC  63065       Geraetenummer kleiner 3? Ja: weiter bei 63065
62970 LDA    #97
62972 STA    185       Sekundaeradresse auf 97 (SAVE auf seriellen Bus) setzen
62974 LDY    183       Laenge des Filenamens = 0?
62976 BNE  62981       Nein: weiter bei 62981
62978 JMP  63248       > I/O-Error #8, "MISSING FILE NAME"

62981 JSR  62421       > OPEN auf seriellen Bus ausgeben
62984 JSR  63119       > Ausgabe "SAVING " und Filenamen
62987 LDA    186       Geraetenummer
62989 JSR  60684       > LISTEN
62992 LDA    185       aktuelle Sekundaeradresse
62994 JSR  60857       > SECOND, Ausgabe Sekundaeradresse
```

SAVE, Speichern von Programmen (Fortsetzung)

```
62997 LDY    #0
62999 JSR  64398      > Programmstartadresse nach (172/173) (Transportzeiger)
63002 LDA    172      Startadresse low
63004 JSR  60893      > CIOUT, Zeichen ausgeben
63007 LDA    173      Startadresse high
63009 JSR  60893      > CIOUT, Zeichen ausgeben
63012 JSR  64721      > Pruefung, ob Ende erreicht?
63015 BCS  63039      Alle Bytes ausgegeben? Ja: weiter bei 63039
63017 LDA   (172),Y   Zeichen aus Speicher lesen
63019 JSR  60893      > CIOUT, Zeichen ausgeben
63022 JSR  65505      > STOP, RUNSTOP-Taste abfragen
63025 BNE  63034      RUNSTOP gedrueckt? Nein: weiter bei 63034

63027 JSR  63042      > CLOSE auf seriellen Bus ausgeben
63030 LDA    #0       Fehlercode Null fuer Abbruch durch RUNSTOP
63032 SEC             Fehlerflag setzen
63033 RTS

63034 JSR  64731      > Transportzeiger erhoehen
63037 BNE  63012      Bei normalem Gebrauch unbedingter Sprung

63039 JSR  60926      > UNLISTEN
```

CLOSE auf seriellen Bus ausgeben

```
63042 BIT    185      aktuelle Sekundaeradresse
63044 BMI  63063      kleiner 128? Nein: fertig
63046 LDA    186      Geraetenummer
63048 JSR  60684      > LISTEN
63051 LDA    185      aktuelle Sekundaeradresse
63053 AND   #239      Bit 4 loeschen
63055 ORA   #224      Bit 5 bis 7 setzen (Kennzeichnung fuer CLOSE)
63057 JSR  60857      > SECOND, Ausgabe Sekundaeradresse
63060 JSR  60926      > UNLISTEN
63063 CLC
63064 RTS

63065 LSR             Bit 0 der Geraetenummer in die Carry-Flag schieben
63066 BCS  63071      Ausgabe auf Recorder? Ja: weiter bei 63071
63068 JMP  63251      > I/O-Error #9, "ILLEGAL DEVICE NUMBER"

63071 JSR  63440      > Pointer auf Cassettenpuffer nach (XR/YR)
63074 BCC  62961      Ist (XR/YR) kleiner 512? Ja: I/O-Error #9
63076 JSR  63544      > "PRESS RECORD & PLAY ON TAPE"
63079 BCS  63118      Abbruch durch RUNSTOP? Ja: RTS
63081 JSR  63119      > Ausgabe "SAVING " und Filenamen
63084 LDX     #3      Kennzeichnung fuer Absolutprogramm
63086 LDA    185      aktuelle Sekundaeradresse
63088 AND     #1      Ist Bit 0 gesetzt?
63090 BNE  63094      Ja: weiter bei 63094
63092 LDX     #1      Kennzeichnung fuer Relativprogramm
63094 TXA
63095 JSR  63338      > Dateikopf in Puffer erzeugen und auf Band schreiben
63098 BCS  63118      Fehler? Ja: RTS
63100 JSR  63591      > Programm auf Band schreiben
63103 BCS  63118      Fehler? Ja: RTS
63105 LDA    185      aktuelle Sekundaeradresse
63107 AND     #2      Ist Bit 1 gesetzt?
63109 BEQ  63117      Nein: fertig
```

SAVE, Speichern von Programmen (Fortsetzung)

```
63111 LDA     #5      Kennzeichnung fuer 'End Of Tape'-Block
63113 JSR  63338      > EOT-Block erzeugen und auf Band schreiben
63116 BIT    ...
63117 CLC
63118 RTS
```

Ausgabe von "SAVING " und Filenamen

```
63119 LDA    157      Ausgabe unterdruecken (Direktmodus)?
63121 BPL  63118      Ja: RTS
63123 LDY    #81      Offset fuer "SAVING "
63125 JSR  61743      > Meldung ausgeben
63128 JMP  62913      > Filenamen ausgeben
```

UDTIM, interne Uhr um eine 60stel Sekunde weitersetzen

```
63131 LDX     #0
63133 INC    162      LSB erhoehen
63135 BNE  63143
63137 INC    161
63139 BNE  63143
63141 INC    160      MSB erhoehen
63143 SEC
63144 LDA    162      momentane Uhrzeit in 60stel Sekunden
63146 SBC     #1      mit 24 Stunden (60stel-Sekunden-Darstellung) vergleichen
63148 LDA    161
63150 SBC    #26
63152 LDA    160
63154 SBC    #79
63156 BCC  63164      24 Stunden erreicht? Nein: weiter bei 63164
63158 STX    160      alle Stellen loeschen
63160 STX    161
63162 STX    162
```

Flag fuer diverse Tasten (zur RUNSTOP-Abfrage) aktualisieren

```
63164 LDA  56321      Tastaturdecoderausgang
63167 CMP  56321      zum Entprellen noch einmal abfragen
63170 BNE  63164
63172 TAX             RUNSTOP gedrueckt?
63173 BMI  63194      Nein: weiter bei 63194
63175 LDX   #189      Bitmuster zur Abfrage der Reihen mit den Shift-Tasten
63177 STX  56320      Tastaturreihenausgang festsetzen
63180 LDX  56321      Tastaturdecoderausgang
63183 CPX  56321      zum Entprellen noch einmal abfragen
63186 BNE  63180
63188 STA  56320
63191 INX             War in keiner der beiden abgefragten Reihen eine Taste
63192 BNE  63196      gedrueckt (speziell Shift)? Nein: kein RUNSTOP
63194 STA    145      Flag fuer diverse Tasten setzen
63196 RTS
```

RDTIM, Uhrzeit lesen

```
63197 SEI
63198 LDA    162
63200 LDX    161
63202 LDY    160
```

SETTIM, Uhrzeit setzen

```
63204 SEI
63205 STA    162
63207 STX    161
63209 STY    160
63211 CLI
63212 RTS
```

STOP: Abfrage der RUNSTOP-Taste

```
63213 LDA    145        Flag fuer diverse Tasten
63215 CMP   #127        RUNSTOP (alleine) gedrueckt?
63217 BNE  63226        Nein: RTS
63219 PHP               Statusflags fuer spaetere Abfrage merken
63220 JSR  65484        > CLRCHN, aktive I/O-Kanaele schliessen
63223 STA    198        Tastaturpufferindex loeschen
63225 PLP               Statusflags wiederherstellen
63226 RTS
```

I/O-Error-Handling, Einsprungstellen fuer Errors

```
63227 LDA     #1        TOO MANY FILES
63229 BIT    ...
63230 LDA     #2        FILE OPEN
63232 BIT    ...
63233 LDA     #3        FILE NOT OPEN
63235 BIT    ...
63236 LDA     #4        FILE NOT FOUND
63238 BIT    ...
63239 LDA     #5        DEVICE NOT PRESENT
63241 BIT    ...
63242 LDA     #6        NOT INPUT FILE
63244 BIT    ...
63245 LDA     #7        NOT OUTPUT FILE
63247 BIT    ...
63248 LDA     #8        MISSING FILE NAME
63250 BIT    ...
63251 LDA     #9        ILLEGAL DEVICE NUMBER
63253 PHA
63254 JSR  65484        > CLRCHN, aktive I/O-Kanaele schliessen
63257 LDY     #0        Offset fuer "I/O ERROR #"
63259 BIT    157        Ausgabe der I/O-Errors unterdruecken?
63261 BVC  63273        Ja: weiter bei 63273
63263 JSR  61743        > Meldung drucken
63266 PLA               Fehlercode holen
63267 PHA               und fuer spaetere Verwendung wieder auf Stack legen
63268 ORA    #48        in ASCII umwandeln
63270 JSR  65490        > CHROUT, Zeichen im Accu drucken
63273 PLA               Fehlercode wiederherstellen
63274 SEC               Fehlerflag setzen
63275 RTS
```

verlangten Dateikopf von Band in Puffer lesen

```
63276 LDA    147        Flag fuer 'VERIFY'
63278 PHA               auf Stack retten
63279 JSR  63553        > Block von Band in Puffer lesen
63282 PLA
63283 STA    147        Flag fuer 'VERIFY' wiederherstellen
63285 BCS  63337        Fehler beim Lesen? Ja: RTS
```

verlangten Dateikopf von Band in Puffer lesen (Fortsetzung)

```
63287 LDY     #0
63289 LDA   (178),Y   erstes Zeichen im Cassettenpuffer
63291 CMP     #5      = 5 (End of Tape)?
63293 BEQ   63337     Ja: RTS
63295 CMP     #1      = 1 (Relativprogramm, meist BASIC)?
63297 BEQ   63307     Ja: weiter bei 63307
63299 CMP     #3      = 3 (Absolutprogramm, kein APPEND an Speicheranfang!)
63301 BEQ   63307     Ja: weiter bei 63307
63303 CMP     #4      = 4 (Kennzeichnung fuer Startblock eines Datenfiles)
63305 BNE   63276     Nein: weitersuchen, zurueck nach 63276
63307 TAX             Kennzeichnung merken
63308 BIT    157      Ausgabe unterdruecken (Direktmodus)?
63310 BPL   63335     Ja: weiter bei 63335
63312 LDY    #99      Offsetpointer fuer "FOUND "
63314 JSR   61743     > Meldung ausgeben
63317 LDY     #5      Index fuer Filenamen
63319 LDA   (178),Y   Zeichen aus Cassettenpuffer lesen
63321 JSR   65490     > CHROUT, Zeichen ausgeben
63324 INY             Index erhoehen
63325 CPY    #21      alle 16 Zeichen des Filenamens ausgegeben?
63327 BNE   63319     Nein: weitermachen ...
63329 LDA    161      mittleres Byte der internen Uhr
63331 JSR   58592     > warten, bis Zeit vorbei oder Taste gedrueckt
63334 NOP
63335 CLC
63336 DEY
63337 RTS
```

Dateikopf erzeugen und auf Band schreiben

```
63338 STA    158      Kennzeichnung fuer Dateityp merken
63340 JSR   63440     > Pointer auf Cassettenpuffer nach (XR/YR)
63343 BCC   63439     Ist (XR/YR) kleiner 512? Ja: RTS
63345 LDA    194      Startadresse high,
63347 PHA
63348 LDA    193      Startadresse low,
63350 PHA
63351 LDA    175      Endadresse high,
63353 PHA
63354 LDA    174      Endadresse low auf den Stack legen
63356 PHA
63357 LDY   #191      Laenge des Cassettenpuffers - 1
63359 LDA    #32      Code fuer "SPACE"
63361 STA   (178),Y   Cassettenpuffer mit Spaces fuellen
63363 DEY
63364 BNE   63361
63366 LDA    158      Kennzeichnung fuer Dateityp
63368 STA   (178),Y   in erste Pufferposition schreiben
63370 INY
63371 LDA    193      ebenso die Startadresse
63373 STA   (178),Y
63375 INY
63376 LDA    194
63378 STA   (178),Y
63380 INY
63381 LDA    174      und die Endadresse
63383 STA   (178),Y
63385 INY
63386 LDA    175
63388 STA   (178),Y
63390 INY
```

Dateikopf erzeugen und auf Band schreiben (Fortsetzung)

```
63391 STY    159       Speicher fuer Index auf Filenamen im Cassettenpuffer
63393 LDY     #0
63395 STY    158       Pointer auf Filenamen festlegen
63397 LDY    158       Wurden alle Zeichen des Filenamens
63399 CPY    183       in den Cassettenpuffer uebertragen?
63401 BEQ   63415      Ja: weiter bei 63415
63403 LDA   (187),Y    Zeichen aus Filenamen
63405 LDY    159
63407 STA   (178),Y    in Cassettenpuffer uebertragen
63409 INC    158       Pointer auf Filenamen erhoehen
63411 INC    159       Pointer auf Filenamen im Cassettenpuffer erhoehen
63413 BNE   63397      Unbedingter Sprung zum Schleifenanfang

63415 JSR   63447      > Start- und Endadresse des Cassettenpuffers festlegen
63418 LDA   #105
63420 STA    171       Dauer des Header-Signals festlegen
63422 JSR   63595      > Cassettenpuffer auf Band schreiben
63425 TAY              eventuellen Fehlercode im YR aufbewahren
63426 PLA
63427 STA    174       Endadresse low,
63429 PLA
63430 STA    175       Endadresse high,
63432 PLA
63433 STA    193       Startadresse low,
63435 PLA
63436 STA    194       Startadresse high wieder vom Stack holen
63438 TYA              eventuellen Fehlercode wiederherstellen
63439 RTS

63440 LDX    178       Pointer auf Startadresse des Cassettenpuffers
63442 LDY    179       nach (XR/YR) uebertragen
63444 CPY     #2       und vergleichen, ob (XR/YR) kleiner 512
63446 RTS

63447 JSR   63440      > Startadresse des Cassettenpuffers nach (XR/YR)
63450 TXA
63451 STA    193       und nach (193/194) speichern
63453 CLC              zur Startadresse 192 addieren,
63454 ADC   #192       ergibt Endadresse+1 des Cassettenpuffers
63456 STA    174       Endadresse des Cassettenpuffers nach (174/175) bringen
63458 TYA
63459 STA    194
63461 ADC     #0
63463 STA    175
63465 RTS
```

Block mit bestimmten Filenamen suchen

```
63466 JSR   63276      > Block in Puffer lesen, Ausgabe "FOUND " und Filenamen
63469 BCS   63500      Abbruch? Ja: RTS
63471 LDY     #5
63473 STY    159       Index fuer gelesenen Filenamen
63475 LDY     #0
63477 STY    158       Index fuer gesuchten Filename
63479 CPY    183       mit Laenge des gesuchten Filenamen vergleichen
63481 BEQ   63499      alle Einzelzeichen bereits verglichen? Ja: fertig
63483 LDA   (187),Y    Zeichen aus gesuchtem Filenamen
63485 LDY    159
63487 CMP   (178),Y    mit Zeichen aus gesuchtem Filenamen vergleichen
63489 BNE   63466      identisch? Nein: naechsten Block lesen
```

Block mit bestimmten Filenamen suchen (Fortsetzung)

```
63491 INC    158
63493 INC    159         ansonsten Indexpointer erhoehen
63495 LDY    158
63497 BNE  63479         und die Filenamen weiter miteinander vergleichen

63499 CLC
63500 RTS

63501 JSR  63440         > Pointer auf Cassettenpuffer nach (XR/YR) bringen
63504 INC    166         Pointer auf Cassettenpuffer (Zeichenzeiger) erhoehen
63506 LDY    166
63508 CPY   #192         Ende des Cassettenpuffers erreicht, alle Zeichen gelesen?
63510 RTS
```

Ausgabe "PRESS PLAY ON TAPE"; warten, bis PLAY-Taste gedrueckt

```
63511 JSR  63534         > Pruefen, ob PLAY-Taste gedrueckt
63514 BEQ  63542         PLAY-Taste gedrueckt? Ja: fertig
63516 LDY    #27         Offset fuer "PRESS PLAY ON TAPE"
63518 JSR  61743         > Meldung ausgeben
63521 JSR  63696         > bei RUNSTOP Rueckkehr zur uebergeordneten Routine
63524 JSR  63534         > Pruefen, ob PLAY-Taste gedrueckt
63527 BNE  63521         PLAY-Taste gedrueckt? Nein: zurueck nach 63521
63529 LDY   #106         Offset fuer "OK"
63531 JMP  61743         > Meldung ausgeben
```

PLAY-Taste abfragen

```
63534 LDA    #16         Maske fuer Bit 4 (PLAY-Taste)
63536 BIT      1         Port abfragen
63538 BNE  63542         PLAY-Taste gedrueckt? Nein: weiter bei 63542
63540 BIT      1         Port nocheinmal abfragen zum Entprellen
63542 CLC
63543 RTS
```

Ausgabe "PRESS RECORD & PLAY ON TAPE"; warten, bis PLAY-Taste gedrueckt

```
63544 JSR  63534         > Pruefen, ob PLAY-Taste gedrueckt
63547 BEQ  63542         PLAY-Taste gedrueckt? Ja: fertig
63549 LDY    #46         Offset fuer "PRESS RECORD & PLAY ON TAPE"
63551 BNE  63518         Unbedingter Sprung
```

Block vom Band in Puffer lesen

```
63553 LDA     #0
63555 STA    144         I/O-Statusbyte und
63557 STA    147         Verify-Flag ruecksetzen
63559 JSR  63447         > Start- und Endadresse fuer Recorderpuffer festsetzen
63562 JSR  63511         > "PRESS PLAY ON TAPE"
63565 BCS  63598         Abbruch durch RUNSTOP? Ja: weiter bei 63708
63567 SEI               Interrupt verhindern
63568 LDA     #0
63570 STA    170         Leseflag := Abtastung
63572 STA    180         Flag fuer TimerA (Read) := disabled
63574 STA    176         Timing-Konstante
63576 STA    158         Korrekturzaehler fuer Pass1
63578 STA    159         Korrekturzaehler fuer Pass2
63580 STA    156         Flag fuer 'Byte empfangen' ruecksetzen
63582 LDA   #144         Bit 4 (negative Flanke auf FLAG)
63584 LDX    #14         Offset fuer Tape-Read (63788)
63586 BNE  63605         Unbedingter Sprung
```

Block von Puffer auf Band schreiben

```
63588 JSR  63447       > Start- und Endadresse fuer Recorderpuffer festsetzen
63591 LDA   #20
63593 STA   171        Laenge des Headers vor Write auf Cassette
63595 JSR  63544       > "PRESS RECORD & PLAY ON TAPE"
63598 BCS  63708       Abbruch durch RUNSTOP? Ja: weiter bei 63708
63600 SEI              Interrupt verhindern
63601 LDA  #130        Bit 2 (Timeout von TimerB)
63603 LDX    #8        Offset fuer Tape-Write (64618)
63605 LDY  #127
63607 STY  56333       alle IRQs sperren
63610 STA  56333       entsprechenden IRQ fuer Schreiben/Lesen freigeben
63613 LDA  56334       Control Register A
63616 ORA   #25        Bits 5 (Timer laden), 4 (One-Shot) und 0 (Start) setzen
63618 STA  56335       und wieder abspeichern
63621 AND  #145        Bitkombination fuer CRA fuer Neustart
63623 STA   674        nach (674) bringen (wird in 63761 wieder gelesen)
63626 JSR  61604       > warten, bis RS-232 Datenuebertragung abgeschlossen ist
63629 LDA  53265       Register #17 des Video-Chips
63632 AND  #239        Bit 4 fuer BLANK SCREEN loeschen und wieder abspeichern
63634 STA  53265       (daher keine Bildschirmausgabe waehrend Tape-I/O)
63637 LDA   788        momentanen IRQ-Vektor in (788/789) nach (671/672) retten
63640 STA   671
63643 LDA   789
63646 STA   672
63649 JSR  64701       > IRQ-Vektor setzen
63652 LDA    #2
63654 STA   190        Anzahl noch zu schreibende/lesende Blocks festsetzen
63656 JSR  64407       > Tape-Initialisierung, Flags und Zaehler setzen
63659 LDA     1
63661 AND   #31
63663 STA     1        Recordermotor einschalten
63665 STA   192        Flag fuer Recorderkontrolle setzen
63667 LDX  #255
63669 LDY  #255
63671 DEY
63672 BNE  63671
63674 DEX
63675 BNE  63669
63677 CLI              Interrupt (Read/Write fuer Cassette) freigeben
63678 LDA   672        geretteter IRQ-Vektor high identisch mit momentanem
63681 CMP   789        IRQ-Vektor high (also alles geschrieben/gelesen)?
63684 CLC              Fehlerflag loeschen
63685 BEQ  63708       Ja: fertig
63687 JSR  63696       > RUNSTOP-Taste abfragen
63690 JSR  63164       > Flag fuer diverse Tasten (RUNSTOP) aktualisieren
63693 JMP  63678       > und weitermachen ...

63696 JSR  65505       > STOP, prueft RUNSTOP-Taste
63699 CLC              Fehlerflag loeschen
63700 BNE  63713       RUNSTOP-Taste gedrueckt? Nein: RTS
63702 JSR  64659       > Ende Recorder-I/O
63705 SEC              Fehlerflag setzen
63706 PLA              Ruecksprungadresse vom Stack entfernen
63707 PLA
63708 LDA    #0        Code fuer Abbruch
63710 STA   672        geretteten IRQ-Vektor high auf null setzen
63713 RTS
```

Cassettensynchronisation vorbereiten

```
63714 STX    177       XR nach (177) speichern
63716 LDA    176       Timing-Konstante
63718 ASL
63719 ASL              mal 4
63720 CLC
63721 ADC    176       plus Timing-Konstante, also mal 5
63723 CLC
63724 ADC    177       plus (177)
63726 STA    177       und wieder dort ablegen
63728 LDA     #0       Accu enthaelt Highbyte fuer Initialisierung von TimerA
63730 BIT    176       Ist (176) groesser 127?
63732 BMI  63735       Ja: weiter bei 63735
63734 ROL              Carry in untere Bitposition des Accus schieben
63735 ASL    177       Wert zur Initialisierung von TimerA vervierfachen
63737 ROL
63738 ASL    177
63740 ROL
63741 TAX              Highbyte merken
63742 LDA  56326       Ist TimerB low
63745 CMP    #22       kleiner 22 (also Veraenderung von TimerB high bis 63755)
63747 BCC  63742       Ja: zurueck nach 63742
63749 ADC    177       Lowbyte fuer Initialisierung addieren
63751 STA  56324       in TimerA low uebertragen
63754 TXA              Highbyte fuer Initialisierung
63755 ADC  56327       zu TimerB high addieren
63758 STA  56325       und in TimerA high schreiben
63761 LDA    674
63764 STA  56334       TimerA starten
63767 STA    676       Flag fuer 'TimerA abgelaufen' ruecksetzen
63770 LDA  56333       Interrupt Control Register
63773 AND    #16       negative Flanke auf FLAG (Tape-Read)?
63775 BEQ  63786       Nein: weiter bei 63786
63777 LDA   #249       Adresse 63786 auf Stack legen
63779 PHA
63780 LDA    #42
63782 PHA
63783 JMP  65347       > Interruptaufruf simulieren

63786 CLI
63787 RTS
```

Interrupt-Routine zum Lesen von Band

```
63788 LDX  56327       TimerB high Counter
63791 LDY   #255
63793 TYA
63794 SBC  56326       TimerB low Counter vom Wert 255 subtrahieren
63797 CPX  56327       Ist TimerB high Counter seit 63788 vermindert worden?
63800 BNE  63788       Ja: zurueck nach 63788
63802 STX    177       TimerB high Counter nach (177) speichern
63804 TAX              vergangene Zeit low seit letzter Flanke ins XR
63805 STY  56326       Latch von TimerB mit Maximalwert (65535) vorbelegen
63808 STY  56327
63811 LDA    #25
63813 STA  56335       Arbeitsmodus fuer TimerB festsetzen und TimerB starten
63816 LDA  56333       Wert des Interrupt Control Registers (Interrupt Flags)
63819 STA    675       nach (675) uebertragen
63822 TYA              TimerB high Counter vom Wert 255 subtrahieren
63823 SBC    177       (Errechnung vergangene Zeit high seit letzter Flanke)
63825 STX    177       vergangene Zeit low nach (177) speichern
```

Interrupt-Routine zum Lesen von Band (Fortsetzung)

```
63827 LSR                (Accu enthaelt vergangene Zeit high)
63828 ROR     177        vergangene Zeit durch 4 dividieren
63830 LSR
63831 ROR     177
63833 LDA     176        Ist Timing-Konstante
63835 CLC
63836 ADC     #60        plus 60
63838 CMP     177        groesser als die Zeit seit letzter Flanke?
63840 BCS   63916        Ja: keine Informationen, weiter bei 63916
63842 LDX     156        Byte empfangen?
63844 BEQ   63849        Nein: weiter bei 63849
63846 JMP   64096        > ansonsten weiter bei 64096

63849 LDX     163        Byte vollstaendig gelesen?
63851 BMI   63880        Ja: weiter bei 63880
63853 LDX      #0        Wert fuer kurzen Impuls
63855 ADC     #48
63857 ADC     176        Accu fuer Abfrage festsetzen
63859 CMP     177        Wurde kurzer Impuls empfangen?
63861 BCS   63891        Ja: weiter bei 63891
63863 INX                Wert fuer langen Impuls
63864 ADC     #38
63866 ADC     176        Accu fuer Abfrage festsetzen
63868 CMP     177        Wurde langer Impuls empfangen?
63870 BCS   63895        Ja: weiter bei 63895
63872 ADC     #44
63874 ADC     176
63876 CMP     177        Ist vergangene Zeit noch laenger (Byte-Header)?
63878 BCC   63883        Nein: weiter bei 63883
63880 JMP   64016        > empfangenes Byte verarbeiten

63883 LDA     180        TimerA freigegeben?
63885 BEQ   63916        Nein: weiter bei 63916
63887 STA     168        Flag fuer 'Read Error' setzen
63889 BNE   63916        Unbedingter Sprung

63891 INC     169        Flag fuer Impulslaengenwechsel erhoehen
63893 BCS   63897        Unbedingter Sprung

63895 DEC     169        Flag fuer Impulslaengenwechsel vermindern
63897 SEC                vom Abfragewert
63898 SBC     #19
63900 SBC     177        19 sowie die vergangene Zeit subtrahieren
63902 ADC     146        und zu Korrekturflag addieren
63904 STA     146        Korrekturflag fuer Timing-Konstante festsetzen
63906 LDA     164        Flag fuer Empfang beider Impulse
63908 EOR      #1        invertieren
63910 STA     164        und wieder abspeichern
63912 BEQ   63957        Beide Impuls empfangen? Ja: weiter bei 63957
63914 STX     215        empfangenes Signal speichern

63916 LDA     180        TimerA freigegeben?
63918 BEQ   63954        Nein: Interrupt abschliessen
63920 LDA     675        Wert des Interrupt Control Registers (siehe 63816)
63923 AND      #1        erfolgte Aufruf der Leseroutine durch TimerA-Interrupt?
63925 BNE   63932        Ja: weiter bei 63932
63927 LDA     676        TimerA abgelaufen?
63930 BNE   63954        Nein: Interrupt abschliessen
63932 LDA      #0
63934 STA     164        Flag fuer Impulszaehlung loeschen
63936 STA     676        Flag fuer 'Timeout TimerA' setzen
```

Interrupt-Routine zum Lesen von Band (Fortsetzung)

```
63939 LDA    163        Byte vollstaendig gelesen?
63941 BPL  63991        Nein: weiter bei 63991
63943 BMI  63880        ansonsten weiter bei 63880

63945 LDX   #166        Wert fuer Initialisierung von TimerA
63947 JSR  63714        > Timing initialisieren
63950 LDA    155        Register fuer Paritybit
63952 BNE  63883        Ungleich null? Ja: Parity Error, weiter bei 63883
63954 JMP  65212        > Interrupt abschliessen

63957 LDA    146        Korrekturflag fuer Timing-Konstante
63959 BEQ  63968        = 0? Ja: weiter bei 63968
63961 BMI  63966        kleiner null? Ja: weiter bei 63966
63963 DEC    176        Timing-Konstante vermindern
63965 BIT    ...
63966 INC    176        Timing-Konstante erhoehen
63968 LDA     #0
63970 STA    146        Korrekturflag fuer Timing-Konstante loeschen
63972 CPX    215        Wert des empfangenen Impulses mit vorherigem vergleichen
63974 BNE  63991        Ungleich? Ja: alles in Ordnung, weiter bei 63991
63976 TXA               wurde kurzer Impuls empfangen?
63977 BNE  63883        Nein: Lesefehler, weiter bei 63883
63979 LDA    169        Flag fuer Impulslaengenwechsel
63981 BMI  63916        negative Werte wegen des folgenden Compares abfangen
63983 CMP    #16        Wurden 16 aufeinanderfolgende kurze Impulse empfangen?
63985 BCC  63916        Nein: weiter bei 63916
63987 STA    150        ansonsten Flag fuer 'EOB empfangen' setzen
63989 BCS  63916        Unbedingter Sprung

63991 TXA               empfangenes Bit
63992 EOR    155        mit Inhalt des Registers fuer das Paritybit verknuepfen
63994 STA    155        und dort wieder abspeichern
63996 LDA    180        TimerA freigegeben?
63998 BEQ  63954        Nein: Interrupt abschliessen
64000 DEC    163        Bitzaehler vermindern
64002 BMI  63945        Paritybit empfangen? Ja: weiter bei 63945
64004 LSR    215        ansonsten gelesenes Bit
64006 ROR    191        in serielles Shift-Register fuer empfangenes Byte bringen
64008 LDX   #218        Wert fuer Initialisierung von TimerA
64010 JSR  63714        > Timing initialisieren
64013 JMP  65212        > Interrupt abschliessen

64016 LDA    150        Wurde EOB empfangen?
64018 BEQ  64024        Nein: weiter bei 64024
64020 LDA    180        TimerA freigegeben?
64022 BEQ  64031        Nein: weiter bei 64031
64024 LDA    163        Ist laufender Bitzaehler
64026 BMI  64031        negativ? Ja: weiter bei 64031
64028 JMP  63895        > kein Byte-Header zu erwarten, langen Impuls verarbeiten

64031 LSR    177        vergangene Zeit seit letzter negativer Flanke halbieren
64033 LDA   #147
64035 SEC
64036 SBC    177        und von 147 subtrahieren
64038 ADC    176        plus der Timing-Konstanten
64040 ASL               Ergebnis verdoppeln
64041 TAX               ergibt Wert fuer die Initialisierung von TimerA
64042 JSR  63714        > Timing initialisieren
64045 INC    156        Flag fuer 'Byte empfangen' setzen
64047 LDA    180        TimerA freigegeben?
64049 BNE  64068        Ja: weiter bei 64068
```

Interrupt-Routine zum Lesen von Band (Fortsetzung)

```
64051 LDA    150      Wurde EOB empfangen?
64053 BEQ  64093      Nein: Interrupt abschliessen
64055 STA    168      Flag fuer Lesefehler setzen
64057 LDA     #0
64059 STA    150      Flag fuer EOB ruecksetzen
64061 LDA   #129
64063 STA  56333      TimerA-Interrupt freigeben
64066 STA    180      Flag fuer 'TimerA enabled' setzen
64068 LDA    150      Flag fuer 'EOB empfangen'
64070 STA    181      in Flag fuer 'gueltiges EOB empfangen' uebertragen
64072 BEQ  64083      kein EOB? Ja: weiter bei 64083
64074 LDA     #0
64076 STA    180      Flag fuer 'TimerA disabled setzen'
64078 LDA     #1
64080 STA  56333      TimerA-Interrupt sperren
64083 LDA    191      Inhalt des seriellen Shift-Registers fuer Read
64085 STA    189      in Register fuer gelesenes Byte bringen
64087 LDA    168      Flag fuer Lesefehler (Parity etc.)
64089 ORA    169      mit Flag fuer Impulslaengenwechsel verknuepfen
64091 STA    182      ergibt Flag fuer Lesefehler des gesamten Bytes
64093 JMP  65212      > Interrupt abschliessen

64096 JSR  64407      > Flags ruecksetzen, Zaehler initialisieren
64099 STA    156      Flag fuer 'Byte empfangen' ruecksetzen
64101 LDX   #218      Wert fuer Initialisierung von TimerA
64103 JSR  63714      > Timing initialisieren
64106 LDA    190      Anzahl noch zu verarbeitende Blocks
64108 BEQ  64112      = 0? Ja: weiter bei 64112
64110 STA    167      Anzahl noch zu lesende Blocks nach (167) uebertragen
64112 LDA    #15      Maskenwert fuer Zaehlung vor Lesen
64114 BIT    170      Flag fuer Lesen von Cassette = Ende?
64116 BPL  64141      (alle erwarteten Zeichen empfangen) Ja: weiter bei 64141
64118 LDA    181      gueltiges EOB empfangen?
64120 BNE  64134      Ja: weiter bei 64134
64122 LDX    190      Anzahl noch zu lesende Blocks
64124 DEX             = 1?
64125 BNE  64138      Nein: Interrupt abschliessen
64127 LDA     #8      Bit 3, 'Long Block'
64129 JSR  65052      > Status setzen
64132 BNE  64138      Unbedingter Sprung zum Interruptabschluss

64134 LDA     #0
64136 STA    170      Flag fuer Lesen von Cassette auf Abtastung setzen
64138 JMP  65212      > Interrupt abschliessen

64141 BVS  64192      Flag fuer Lesen von Tape = Lesen? Ja: weiter bei 64192
64143 BNE  64169      = Zaehlung? Ja: weiter bei 64169
64145 LDA    181      EOB empfangen?
64147 BNE  64138      Ja: Interrupt abschliessen
64149 LDA    182      Byte-Lesefehler?
64151 BNE  64138      Ja: Interrupt abschliessen
64153 LDA    167      Anzahl noch zu lesende Blocks
64155 LSR             Bit 0 in Carry schieben
64156 LDA    189      gelesenes Byte
64158 BMI  64163      Bit 7 gesetzt (Zaehlungsbyte)? Ja: weiter bei 64163
64160 BCC  64186      noch zu lesende Blocks = 1? Nein: weiter bei 64186
64162 CLC
64163 BCS  64186      noch zu lesende Blocks = 1? Ja: weiter bei 64186
64165 AND    #15      Bits 0 bis 3 isolieren
64167 STA    170      und fuer Zaehlung abspeichern
```

Interrupt-Routine zum Lesen von Band (Fortsetzung)

```
64169 DEC    170        Zaehler vermindern; alle Synchronisationsbytes empfangen?
64171 BNE   64138       Nein: Interrupt abschliessen
64173 LDA    #64
64175 STA    170        Leseflag := Lesen
64177 JSR   64398       > Programmstartadresse in Ladezeiger uebertragen
64180 LDA     #0
64182 STA    171        Pruefsummenwort loeschen
64184 BEQ   64138       Abschluss IRQ

64186 LDA   #128
64188 STA    170        Leseflag := Ende
64190 BNE   64138       Abschluss IRQ
64192 LDA    181        End Of Block (EOB) empfangen?
64194 BEQ   64206       Nein: weiter bei 64206
64196 LDA     #4        Bit 2 (Short Block)
64198 JSR   65052       > Status setzen
64201 LDA     #0        Leseflag := Abtastung
64203 JMP   64330       > weitermachen ...

64206 JSR   64721       > Ende erreicht?
64209 BCC   64214       Nein: weitermachen ...
64211 JMP   64328       > Ende Read (Block)

64214 LDX    167        noch zu lesende Blocks
64216 DEX               = 1?
64217 BEQ   64264       Ja: Pass2, Korrekturpass
64219 LDA    147        Flag fuer 'VERIFY' gesetzt?
64221 BEQ   64235       Nein: weiter bei 64235
64223 LDY     #0        Index := 0
64225 LDA    189        gelesenes Byte
64227 CMP   (172),Y     mit dem im Speicher stehenden vergleichen
64229 BEQ   64235       Gleich: weiter bei 64235
64231 LDA     #1
64233 STA    182        Flag fuer Zeichen-Lesefehler setzen
64235 LDA    182        Fehler aufgetreten?
64237 BEQ   64314       Nein: weiter bei 64314
64239 LDX    #61        bereits 31 Fehler
64241 CPX    158        aufgetreten?
64243 BCC   64307       Ja: Bit 4 (nicht korrigierbarer Fehler) im Status setzen
64245 LDX    158        Index fuer Lesefehler
64247 LDA    173        laufendes Adressbyte high
64249 STA    257,X      im Stack speichern
64252 LDA    172        laufendes Adressbyte low
64254 STA    256,X      ebenso
64257 INX
64258 INX               Index um zwei erhoehen
64259 STX    158        und abspeichern
64261 JMP   64314       > weiter bei 64314

64264 LDX    159        bereits alle Lesefehler
64266 CPX    158        korrigiert?
64268 BEQ   64323       Ja: weiter bei 64385
64270 LDA    172        laufendes Adressbyte low
64272 CMP    256,X      mit fehlerhaftem Adressbyte low vergleichen
64275 BNE   64323       Ungleich: weiter bei 64323
64277 LDA    173        laufendes Adressbyte high
64279 CMP    257,X      mit fehlerhaftem Adressbyte high vergleichen
64282 BNE   64323       Ungleich: weiter bei 64323
64284 INC    159
64286 INC    159        Korrekturzaehler Pass2 um zwei erhoehen
```

Interrupt-Routine zum Lesen von Band (Fortsetzung)

```
64288 LDA    147       Verify-Flag gesetzt?
64290 BEQ  64303       Nein: weiter bei 64303
64292 LDA    189       gelesenes Byte
64294 LDY     #0       Index := 0
64296 CMP   (172),Y    mit dem im Speicher stehenden Byte vergleichen
64298 BEQ  64323       Gleich: weiter bei 64323
64300 INY
64301 STY    182       Fehlerflag setzen
64303 LDA    182       Fehler aufgetreten?
64305 BEQ  64314       Nein: weiter bei 64314
64307 LDA    #16       Eit 4 (Lesefehler, VERIFY-ERROR)
64309 JSR  65052       > Status setzen
64312 BNE  64323       Unbedingter Sprung

64314 LDA    147       Verify-Flag gesetzt?
64316 BNE  64323       Ja: weiter bei 64323
64318 TAY              Index := 0
64319 LDA    189       gelesenes Byte
64321 STA   (172),Y    abspeichern
64323 JSR  64731       > Pointer erhoehen
64326 BNE  64395       Abschluss IRQ

64328 LDA   #128
64330 STA    170       Leseflag := Ende
64332 SEI
64333 LDX     #1
64335 STX  56333       TimerA-Interrupt verhindern
64338 LDX  56333       Interrupt-Flags ruecksetzen
64341 LDX    190       Anzahl noch zu verarbeitende Blocks
64343 DEX              = 0?
64344 BMI  64348       Ja: weiter bei 64348
64346 STX    190       neuen Wert speichern
64348 DEC    167       Blockzaehler vermindern
64350 BEQ  64360       = 0? Ja: weiter bei 64360
64352 LDA    158       Fehler in Pass1 aufgetreten?
64354 BNE  64395       Ja: weiter bei 64395
64356 STA    190       Anzahl noch zu verarbeitende Blocks := 0
64358 BEQ  64395       Abschluss IRQ

64360 JSR  64659       > Ende Tape I/O
64363 JSR  64398       > Programmstartadresse in Ladezeiger uebertragen
64366 LDY     #0
64368 STY    171       Pruefsumme loeschen
64370 LDA   (172),Y    Pruefsumme bilden
64372 EOR    171
64374 STA    171
64376 JSR  64731       > Transportzeiger erhoehen
64379 JSR  64721       > Ende erreicht?
64382 BCC  64370       Nein: weitermachen ...
64384 LDA    171       Pruefsumme mit
64386 EOR    189       gelesener Pruefsumme vergleichen
64388 BEQ  64395       Gleich: weiter bei 64395
64390 LDA    #32       Bit 5 (Pruefsummenfehler)
64392 JSR  65052       > Status setzen
64395 JMP  65212       > Abschluss IRQ
```

Programmstartadresse in Transportzeiger uebertragen

```
64398 LDA    194       Programmstartadresse high
64400 STA    173       in Transportzeiger bringen
64402 LDA    193       Programmstartadresse low
64404 STA    172       in Transportzeiger bringen
64406 RTS
```

Vorbereitung fuer Byte-Read/Write

```
64407 LDA     #8
64409 STA    163       8-Bit-Zaehlung vorbereiten
64411 LDA     #0
64413 STA    164       Bit-Impuls-Flag
64415 STA    168       Lesefehler
64417 STA    155       Parity-Bit
64419 STA    169       Flag fuer Impulslaengenwechsel
64421 RTS
```

Impuls auf Band schreiben

```
64422 LDA    189       serielles Bit-Shift Register
64424 LSR              Bit 0 in die Carry-Flag schieben
64425 LDA    #96       Wert fuer "short" (kurzer Impuls)
64427 BCC  64431       Carry = 0? Ja: weiter bei 64431
64429 LDA   #176       Wert fuer "lòng" (langer Impuls)
64431 LDX     #0       Timer-Wert high
64433 STA  56326       Accu in TimerB low Latch
64436 STX  56327       XR in TimerB high Latch
64439 LDA  56333       Interruptflags ruecksetzen
64442 LDA    #25
64444 STA  56335       TimerB starten
64447 LDA      1       Tape-Write Bit
64449 EOR     #8       invertieren,
64451 STA      1       da Impuls als Vollwelle gespeichert wird
64453 AND     #8       momentanen Wert in Zero-Flag
64455 RTS
```

Flag fuer 'Block geschrieben' setzen

```
64456 SEC
64457 ROR    182       Flag fuer 'Block geschrieben' negativ machen
64459 BMI  64521       Abschluss IRQ
```

Interrupt-Routine fuer Schreiben auf Band (Puffer)

```
64461 LDA    168       "byte"-Impuls geschrieben?
64463 BNE  64483       Ja: weiter bei 64483
64465 LDA    #16       Wert low,
64467 LDX     #1       Wert high fuer 'byte'-Frequenz
64469 JSR  64433       > "byte"-Impuls auf Band
64472 BNE  64521       falls noch erste Halbwelle, Abschluss IRQ
64474 INC    168       Flag fuer '"byte"-Impuls geschrieben' setzen
64476 LDA    182       Flag fuer 'Block geschrieben' positiv? (vgl. 64560 ff)
64478 BPL  64521       Ja: Abschluss IRQ
64480 JMP  64599       > Fortsetzung Write, wenn Block geschrieben

64483 LDA    169       "long"-Impuls geschrieben?
64485 BNE  64496       Ja: weiter bei 64496
64487 JSR  64429       > "long"-Impuls auf Band
64490 BNE  64521       falls noch erste Halbwelle, Abschluss IRQ
64492 INC    169       Flag fuer '"long"-Impuls geschrieben' setzen
64494 BNE  64521       Abschluss IRQ
```

Interrupt-Routine fuer Schreiben auf Band (Fortsetzung)

```
64496 JSR   64422        > Bit auf Band
64499 BNE   64521        falls noch erste Halbwelle, Abschluss IRQ
64501 LDA     164        Bit Impuls Flag
64503 EOR      #1        invertieren
64505 STA     164        und speichern
64507 BEQ   64524        wenn 0, dann beide Bit-Impulse geschrieben
64509 LDA     189        Bit 0 im Bit-Shift-Register invertieren
64511 EOR      #1        logisch 0 wird als "short/long",
64513 STA     189        logisch 1 als "long/short" gespeichert
64515 AND      #1        lfd. Bit herausfiltern
64517 EOR     155        mit 'Parity-Bit fuer Byte' verknuepfen
64519 STA     155        und dort wieder abspeichern
64521 JMP   65212        > Abschluss IRQ

64524 LSR     189        Bit herausshiften
64526 DEC     163        8-Bit Zaehler vermindern
64528 LDA     163        Ende erreicht?
64530 BEQ   64590        Ja: Pruefsumme bilden
64532 BPL   64521        Nein: weitermachen, Abschluss IRQ
64534 JSR   64407        > 8-Bit Zaehler setzen, Flags etc. loeschen
64537 CLI
64538 LDA     165        Synchronisationsbytes geschrieben?
64540 BEQ   64560        Ja: weiter bei 64560
64542 LDX      #0
64544 STX     215        Puffer-Pruefsumme loeschen
64546 DEC     165        Synchronisationszaehler vermindern
64548 LDX     190        Anzahl noch zu schreibende Blocks
64550 CPX      #2        = 2 (erster Block geschrieben)?
64552 BNE   64556        Nein: weiter bei 64556
64554 ORA    #128        Bit 7 im Synchronisationsbyte setzen
64556 STA     189        in Bit-Shift-Register bringen
64558 BNE   64521        Abschluss IRQ

64560 JSR   64721        > lfd. Adresse mit Endwert vergleichen
64563 BCC   64575        Kleiner: weiterschreiben
64565 BNE   64456        Ungleich: Flag fuer 'Block geschrieben' setzen
64567 INC     173        sonst lfd. Adresse (high) ungleich machen
64569 LDA     215        Puffer-Pruefsumme
64571 STA     189        in Bit-Shift-Register bringen
64573 BCS   64521        Abschluss IRQ

64575 LDY      #0        Index := 0
64577 LDA   (172),Y      laufendes Zeichen im Speicher
64579 STA     189        in Bit-Shift-Register bringen
64581 EOR     215        Pruefsumme
64583 STA     215        bilden
64585 JSR   64731        > Pointer erhoehen
64588 BNE   64521        Abschluss IRQ

64590 LDA     155        Parity-Bit fuer Byte
64592 EOR      #1        invertieren
64594 STA     189        und ins Bit-Shift-Register bringen
64596 JMP   65212        > Abschluss IRQ
```

Fortsetzung der Write-Routine, falls Block geschrieben

```
64599 DEC   190      noch zu schreibende Blocks
64601 BNE  64606     = 0? Nein: weiter bei 64606
64603 JSR  64714     > Recordermotor ausschalten
64606 LDA   #80
64608 STA   167      Zaehler fuer Laenge der Shorts
64610 LDX    #8      Offset fuer IRQ #1 fuer Write (64618)
64612 SEI
64613 JSR  64701     > IRQ-Vektor setzen
64616 BNE  64596     Abschluss IRQ
```

IRQ #1, Schreiben des Headers

```
64618 LDA  #120      "header"-Impuls
64620 JSR  64431     > schreiben
64623 BNE  64596     falls noch erste Halbwelle, Abschluss IRQ
64625 DEC   167      Zaehler fuer Header vermindern
64627 BNE  64596     Ende erreicht? Nein: Abschluss IRQ
64629 JSR  64407     > 8-Bit-Zaehler setzen, Flags etc. loeschen
64632 DEC   171      Dauer der Shorts vor und nach Daten
64634 BPL  64596     Ende erreicht? Nein: Abschluss IRQ
64636 LDX   #10      IRQ #2 fuer Write (64461)
64638 JSR  64701     > IRQ-Vektor setzen
64641 CLI
64642 INC   171      Dauer der Shorts
64644 LDA   190      alle Blocks geschrieben?
64646 BEQ  64696     Ja: weiter bei 64696
64648 JSR  64398     > Zeiger auf letztes Zeichen setzen
64651 LDX    #9
64653 STX   165      Synchronisationszaehler setzen
64655 STX   182      Flag fuer 'Block geschrieben' ruecksetzen
64657 BNE  64534     Unbedingter Sprung
```

Ende-Recorder-I/O

```
64659 PHP            Status retten
64660 SEI            IRQ verhindern
64661 LDA  53265
64664 ORA   #16
64666 STA  53265     Bildschirminhalt wieder sichtbar machen
64669 JSR  64714     > Recordermotor ausschalten
64672 LDA  #127
64674 STA  56333     alle CIA 6526-IRQs sperren
64677 JSR  64989     > TimerA Interrupt fuer Tastaturabfrage setzen
64680 LDA   672      Ist Highbyte des geretteten IRQs
64683 BEQ  64694     = 0? Ja: fertig
64685 STA   789      geretteten IRQ-Vektor in IRQ-Vektor uebertragen
64688 LDA   671
64691 STA   788
64694 PLP            Status wiederherstellen
64695 RTS

64696 JSR  64659     > Ende Recorder-I/O
64699 BEQ  64596     Abschluss IRQ
```

IRQ-Vektor setzen

```
64701 LDA  64915,X   Wert aus Tabelle in
64704 STA   788      IRQ-Vektor bringen
64707 LDA  64916,X
64710 STA   789
64713 RTS
```

Hilfsroutinen zur Bedienung des Recorders

```
64714 LDA      1       Recordermotor ausschalten
64716 ORA    #32
64718 STA      1
64720 RTS

64721 SEC              Transportzeiger mit Endadresse vergleichen
64722 LDA    172
64724 SBC    174
64726 LDA    173
64728 SBC    175
64730 RTS

64731 INC    172       Transportzeiger erhoehen
64733 BNE  64737
64735 INC    173
64737 RTS
```

RESET-Routine, Aufruf beim Einschalten

```
64738 LDX   #255
64740 SEI
64741 TXS              Stackpointer initialisieren
64742 CLD
64743 JSR  64770       > Cartridge eingesetzt?
64746 BNE  64751       Nein: normaler RESET
64748 JMP (32768)      > Sprung in 8K Cartridge ROM Area

64751 STX  53270       Reset-Bit ruecksetzen
64754 JSR  64931       > I/O-Reset
64757 JSR  64848       > freien Speicherplatz bestimmen
64760 JSR  64789       > Vektoren von 788 bis 819 festlegen
64763 JSR  65371       > Fernsehnorm/Taktfrequenz feststellen
64766 CLI
64767 JMP (40960)      > Sprung nach 58260, Fortsetzung RESET-Routine
```

Test, ob 8K Cartridge ROM Area (32768 bis 40959) belegt

```
64770 LDX     #5       Bereich
64772 LDA  64783,X     von 64784 bis 64788 mit dem
64775 CMP  32771,X     von 32772 bis 32776 vergleichen.
64778 BNE  64783       ungleich? Ja: RTS
64780 DEX
64781 BNE  64772       alle 5 Bytes pruefen
64783 RTS
```

Pruefbytes fuer 64770

```
64784 195 194 205  56  48   'CBM80'
```

RESTOR: Vektoren initialisieren

```
64789 LDX    #48       (Accu/YR) := 64816, Startadresse der Standardvektoren
64791 LDY   #253
64793 CLC              Flag fuer 'Tabelle nach Vektoren' setzen
```

VECTOR: Lesen und Setzen der Sprungvektoren

```
64794 STX     195       Pointer setzen
64796 STY     196
64798 LDY     #31       Zaehler fuer 16 Vektoren setzen
64800 LDA     788,Y     Wert aus Vektortabelle lesen
64803 BCS   64807       Tabellenwert in anderen Bereich kopieren? Ja: ->
64805 LDA    (195),Y
64807 STA    (195),Y
64809 STA     788,Y     Wert in Vektortabelle uebertragen
64812 DEY
64813 BPL   64800
64815 RTS

64816  49 234   59953   IRQ-Vektor
64818 102 254   65126   BRK-Vektor
64820  71 254   65095   NMI-Vektor
64822  74 243   62282   OPEN
64824 145 242   62097   CLOSE
64826  14 242   61966   CHKIN
64828  80 242   62032   CHKOUT
64830  51 243   62259   CLRCHN
64832  87 241   61783   CHRIN
64834 202 241   61898   CHROUT
64836 237 246   63213   STOP
64838  62 241   61758   GETIN
64840  47 243   62255   CLALL
64842 102 254   65126
64844 165 244   62629   LOAD-Vektor
64846 237 245   62957   SAVE-Vektor
```

Pruefung auf freien BASIC-RAM Bereich

```
64848 LDA      #0
64850 TAY
64851 STA       2,Y     Bereich (2,...,257), (512,...,1023) mit 0 fuellen
64854 STA     512,Y
64857 STA     768,Y
64860 INY
64861 BNE   64851
64863 LDX     #60       (XR/YR) := 828, Startadresse des Cassettenpuffers
64865 LDY      #3
64867 STX     178       in Pointer auf Cassettenpuffer bringen
64869 STY     179
64871 TAY               YR := 0
64872 LDA      #3       Startadresse high - 1 fuer RAM-Test
64874 STA     194
64876 INC     194       Pointer high um eins erhoehen
64878 LDA    (193),Y
64880 TAX               vorherigen Wert merken
64881 LDA     #85       Bitmuster 01010101
64883 STA    (193),Y    in Speicherstelle schreiben
64885 CMP    (193),Y    und vergleichen, ob abgespeicherter Wert darin
64887 BNE   64904       Nein: kein RAM mehr, weiter bei 64904
64889 ROL               sollte wohl ASL sein, ergibt aber 10101011
64890 STA    (193),Y    in Speicherstelle schreiben
64892 CMP    (193),Y    und vergleichen, ob abgespeicherter Wert darin
64894 BNE   64904       Nein: kein RAM mehr, weiter bei 64904
64896 TXA
64897 STA    (193),Y    gemerkten Wert wiederherstellen
64899 INY               Index erhoehen, Pageende erreicht?
64900 BNE   64878       Nein: weitermachen ...
64902 BEQ   64876       sonst auch Pointer high um eins erhoehen
```

Pruefung auf freien BASIC-RAM Bereich (Fortsetzung)

```
64904 TYA              Pointer low
64905 TAX              ins XR
64906 LDY    194       Pointer high ins YR
64908 CLC
64909 JSR  65069       > Pointer auf Ende des Arbeitsspeichers setzen
64912 LDA     #8
64914 STA    642       Startadresse high des Speicheranfangs (MEMBOT)
64917 LDA     #4
64919 STA    648       Startadresse high des Bildschirm-RAMs
64922 RTS
```

Tabelle der IRQ-Vektoren fuer Recorder-Bedienung

```
64923 106 252   64618     Write #1 auf Cassette (Header)
64925 205 251   64461     Write #2 auf Cassette (Puffer)
64927  49 234   59953     Normalwert (Tastaturabfrage etc.)
64929  44 249   63788     Read von Cassette
```

I/O-RESET

```
64931 LDA   #127
64933 STA  56333       alle IRQs
64936 STA  56589       und alle NMIs sperren
64939 STA  56320
64942 LDA     #8
64944 STA  56334       Control Register A der IRQ-CIA
64947 STA  56590       Control Register A der NMI-CIA
64950 STA  56335       Control Register B der IRQ-CIA
64953 STA  56591       Control Register b der NMI-CIA
64956 LDX     #0
64958 STX  56323       gesamten Port B der IRQ-CIA auf Eingang setzen (Keyboard)
64961 STX  56579       ebenso Port B der NMI-CIA (User-Port)
64964 STX  54296       Lautstaerke des Synthesizer-Chips auf null setzen
64967 DEX
64968 STX  56322       gesamten Port A der IRQ-CIA auf Ausgang setzen (Keyboard)
64971 LDA     #7
64973 STA  56576       Port A (NMI-CIA)
64976 LDA    #63       Bit 0 bis Bit 5 von Port A (NMI-CIA) auf Ausgang setzen,
64978 STA  56578       Bit 6 und Bit 7 auf Eingang
64981 LDA   #231       Bitmuster 11100111
64983 STA      1       in 6510-Datenport schreiben
64985 LDA    #47       Bitmuster 00101111
64987 STA      0       in 6510-Datenrichtungsregister schreiben
64989 LDA    678       Flag fuer Taktfrequenz (Fernsehnormabhaengig)
64992 BEQ  65004       Fernsehnorm = PAL? Nein: weiter bei 65004
64994 LDA    #37       Wert 16421 fuer Interrupt nach Timer A
64996 STA  56324
64999 LDA    #64
65001 JMP  65011
65004 LDA   #149       Wert 17045 fuer Interrupt nach Timer A
65006 STA  56324
65009 LDA    #66
65011 STA  56325
65014 JMP  65390       > weiter bei 65390
```

SETNAM: Festsetzung der Daten fuer den Filenamen

```
65017 STA    183       Laenge des Filenamens
65019 STX    187       Startadresse des Filenamens low,
65021 STY    188       high
65023 RTS
```

SETLFS: Festsetzung der OPEN-Parameter

```
65024 STA    184       Filenummer
65026 STX    186       Geraetenummer
65028 STY    185       Sekundaeradresse
65030 RTS
```

READST: I/O-Status in Accu bringen

```
65031 LDA    186       Geraetenummer
65033 CMP     #2       =2 (RS-232 Kanal)?
65035 BNE  65050       Nein: Bus/Tape-Status, weiter bei 65050
65037 LDA    663       RS-232 Statusbyte
65040 PHA              merken,
65041 LDA     #0
65043 STA    663       loeschen,
65046 PLA              alten Status in Accu
65047 RTS
```

SETMSG: Ausgabemodus fuer Betriebssystemmeldungen setzen

```
65048 STA    157       Ausgabeflag, Flag fuer Direktmodus
65050 LDA    144       Statusbyte lesen
```

Einsprung fuer 'Status setzen'

```
65052 ORA    144       im Accu gesetzte Bits im
65054 STA    144       Statusregister setzen
65056 RTS
```

SETTMO, timeout-disable loeschen (Accu=0), setzen (Accu=128)

```
65057 STA    645       timeout-disable
65060 RTS
```

MEMTOP, Lesen (Carry=1) und Setzen (Carry=0) des Speicherendes

```
65061 BCC  65069       Carry gesetzt? Nein: weiter bei 65069
65063 LDX    643       Speicherende low ins XR,
65066 LDY    644       high ins YR
65069 STX    643       XR nach Speicherende low,
65072 STY    644       YR nach high speichern
65075 RTS
```

MEMBOT, Lesen (Carry=1) und Setzen (Carry=0) des Speicherbeginns

```
65076 BCC  65084       Carry gesetzt? Nein: weiter bei 65084
65078 LDX    641       Speicherbeginn low ins XR,
65081 LDY    642       high ins YR
65084 STX    641       XR nach Speicherende low,
65087 STY    642       YR nach high speichern
65090 RTS
```

NMI-Routine, Aufruf durch RESTORE-Taste

```
65091 SEI
65092 JMP   (792)     Normalwert des Vektors (792/793): 65095
65095 PHA             Register retten
65096 TXA
65097 PHA
65098 TYA
65099 PHA
65100 LDA   #127
65102 STA  56589      alle NMIs verhindern (Interrupt Control Register)
65105 LDY  56589      Aufruf durch RESTORE-Taste?
65108 BMI  65138      Nein: weiter bei 65138
65110 JSR  64770      > Cartridge eingesetzt?
65113 BNE  65118      Nein: normale NMI-Routine
65115 JMP (32770)     > Sprung in 8K Cartridge ROM Area, Warmstart-Vektor
```

NMI-Routine, Aufruf durch RESTORE-Taste (Fortsetzung)

```
65118 JSR  63164      > Flag fuer diverse Tasten (RUNSTOP) aktualisieren
65121 JSR  65505      > STOP-Taste gedrueckt?
65124 BNE  65138      Nein: weiter bei 65138
65126 JSR  64789      > RESTOR: Vektortabelle (788,...,819) initialisieren
65129 JSR  64931      > I/O-Reset
65132 JSR  58648      > Screen Editor Reset
65135 JMP (40962)     > 58235: BASIC-Warmstart
```

NMI-Handling fuer RS-232

```
65138 TYA             Wert aus dem ICR
65139 AND    673      mit dem RS-232 Register fuer aktive NMIs verknuepfen
65142 TAX             und ins XR retten
65143 AND     #1      Werden Daten ueber den RS-232 Kanal gesendet?
65145 BEQ  65187      Nein: weiter bei 65187
65147 LDA  56576      Port A der NMI-CIA
65150 AND   #251      Bit 2 fuer RS-232-OUT (fuer ORA) auf null setzen
65152 ORA    181      zu sendendes Bit in den Wert fuer Port A hineinbringen
65154 STA  56576      und wieder in Port A abspeichern
65157 LDA    673      Wert aus RS-232 Register fuer aktive NMIs
65160 STA  56589      in Interrupt Control Register (NMI-CIA) uebertragen
65163 TXA             aktive NMIs fuer RS-232 Datentransfer
65164 AND    #18      Bits 1 und 4 isolieren (NMIs fuer Datenempfang)
65166 BEQ  65181      Beide Bits geloescht? Ja: weiter bei 65181
65168 AND     #2      Aufruf der NMI-Routine durch Timeout von TimerB?
65170 BEQ  65178      Nein: Startbit, weiter bei 65178
65172 JSR  65238      > empfangenes Bit verarbeiten
65175 JMP  65181      > weiter bei 65181

65178 JSR  65287      > Datenempfang fuer Byte neu vorbereiten
65181 JSR  61115      > Uebertragung des naechsten Bits vorbereiten
65184 JMP  65206      > Abschluss RS-232 NMI-Handling

65187 TXA
65188 AND     #2      werden Daten empfangen?
65190 BEQ  65198      Nein: weiter bei 65198
65192 JSR  65238      > empfangenes Bit verarbeiten
65195 JMP  65206      > Abschluss RS-232 NMI-Handling

65198 TXA
65199 AND    #16      wird auf Startbit gewartet?
65201 BEQ  65206      Nein: Abschluss RS-232-NMI
65203 JSR  65287      > Datenempfang fuer Byte neu vorbereiten
```

RS-232-NMI-Handling abschliessen

```
65206 LDA    673       Wert aus RS-232 Register fuer aktive MMIs
65209 STA  56589       in Interrupt Control Register (NMI-CIA) uebertragen

65212 PLA              Register wiederherstellen
65213 TAY
65214 PLA
65215 TAX
65216 PLA
65217 RTI
```

Timerkonstantentabelle fuer NTSC (vgl. 58604, 62508 ff)

```
65218 193  39      50   Baud
65220  62  26      75   Baud
65222 197  17     110   Baud
65224 116  14     134.5 Baud
65226 237  12     150   Baud
65228  69   6     300   Baud
65230 240   2     600   Baud
65232  70   1    1200   Baud
65234 184   0    1800   Baud
65236 113   0    2400   Baud

65238 LDA  56577       Port B (NMI-CIA) abfragen
65241 AND     #1       Bit 0 (RS-232 IN) isolieren
65243 STA    167       und in Register fuer empfangenes Bit bringen
65245 LDA  56582       TimerB fuer Datenempfang neu anpassen
65248 SBC    #28
65250 ADC    665
65253 STA  56582
65256 LDA  56583
65259 ADC    666
65262 STA  56583
65265 LDA    #17
65267 STA  56591       Arbeitsmodus fuer TimerB festsetzen und Timer starten
65270 LDA    673       Wert aus RS-232 Register fuer aktive NMIs
65273 STA  56589       im Interrupt Control Register (NMI-CIA) uebertragen
65276 LDA   #255
65278 STA  56582       Latch von TimerB mit Maximalwert vorbelegen
65281 STA  56583
65284 JMP  61273       > weiter bei 61273

65287 LDA    661       TimerB fuer Baud Rate setzen
65290 STA  56582
65293 LDA    662
65296 STA  56583
65299 LDA    #17
65301 STA  56591       Arbeitsmodus fuer TimerB festsetzen und Timer starten
65304 LDA    #18       Bits 1 und 4
65306 EOR    673       mit RS-232 Register fuer aktive NMIs verknuepfen
65309 STA    673       und wieder abspeichern
65312 LDA   #255
65314 STA  56582       Latch von TimerB mit Maximalwert vorbelegen
65317 STA  56583
65320 LDX    664       Register Wortlaenge
65323 STX    168       in Zaehler fuer Wortlaenge uebertragen
65325 RTS
```

Routine fuer RS-232 Handling

```
65326 TAX                  Werte aus Tabelle fuer Baud Rate
65327 LDA    662           in Werte fuer die Baud Rate zum Senden umwandeln
65330 ROL                  (Aufruf von 62538)
65331 TAY
65332 TXA
65333 ADC   #200
65335 STA    665
65338 TYA
65339 ADC     #0
65341 STA    666
65344 RTS

65345 NOP                  Aufruf von 63783
65346 NOP
65347 PHP                  Statusregister auf Stack legen,
65348 PLA                  in Accu bringen
65349 AND   #239           Bit 4 (Break Flag) loeschen
65351 PHA                  und als Statusregisterinhalt auf Stack legen
```

IRQ-Routine

```
65352 PHA                  Register retten
65353 TXA
65354 PHA
65355 TYA
65356 PHA
65357 TSX
65358 LDA    260,X         Processor-Status-Register in Accu bringen
65361 AND    #16           BRK-Flag isolieren
65363 BEQ  65368           gesetzt? Nein: IRQ, weiter bei 65368
65365 JMP   (790)          =65126: Sprung ueber BRK-Vektor
65368 JMP   (788)          =59953: Sprung ueber IRQ-Vektor
```

Feststellung der TV-Norm und der davon abhaengigen Taktfrequenz

```
65371 JSR  58648           > Screen Editor Reset
65374 LDA  53266           Raster Register des Video-Chips
65377 BNE  65374           warten, bis Inhalt des Raster Registers = 0
65379 LDA  53273           Raster Count = Latched Raster Count (Raster Compare IRQ)?
65382 AND     #1
65384 STA    678           Bit 0 als Flag speichern
65387 JMP  64989           > Werte fuer Interrupt-Timer festsetzen

65390 LDA   #129
65392 STA  56333           TimerA-IRQ (fuer Tastaturabfrage) freigeben
65395 LDA  56334
65398 AND   #128
65400 ORA    #17
65402 STA  56334           TimerA laden und starten (dauernd)
65405 JMP  61070           > Clock Out := high

65408   0                  ?
```

Sprungtabelle fuer KERNAL-Routinen

bei Vektoren Angabe des Normalwerts beim Einschalten (ohne Erweiterungen)

```
65409 JMP  65371      Feststellung der TV-Norm (PAL, NTSC) und Taktfrequenz
65412 JMP  64931      I/O-Reset
65415 JMP  64848      Pruefung auf freien BASIC-RAM-Bereich
65418 JMP  64789      RESTOR: Vektoren initialisieren
65421 JMP  64794      VECTOR: Lesen und Setzen der Vektoren
65424 JMP  65048      SETMSG: Ausgabemodus setzen
65427 JMP  60857      SECOND: Ausgabe Sekundaeradresse nach LISTEN
65430 JMP  60871      TKSA:   Ausgabe Sekundaeradresse nach TALK
65433 JMP  65061      MEMTOP: Lesen und Setzen des Speicherendes
65436 JMP  65076      MEMBOT: Lesen und Setzen des Speicheranfangs
65439 JMP  60039      SCNKEY: Tastaturabfrage
65442 JMP  65057      SETTMO: Time-Out-Flag setzen
65445 JMP  60947      ACPTR:  Zeichen vom IEEE-Bus in Accu
65448 JMP  60893      CIOUT:  Ausgabe Accu auf IEEE-Bus
65451 JMP  60911      UNTLK:  Untalk auf IEEE-Bus
65454 JMP  60926      UNLSN:  Unlisten auf IEEE-Bus
65457 JMP  60684      LISTEN: an IEEE-Geraet (Nummer im Accu)
65460 JMP  60681      TALK:   an IEEE-Geraet (Nummer im Accu)
65463 JMP  65031      READST: I/O-Status in Accu
65466 JMP  65024      SETLFS: Festsetzung Parameter fuer OPEN
65469 JMP  65017      SETNAM: Festsetzung Filename
65472 JMP   (794)     =62282: OPEN:   spezifiziertes File oeffnen
65475 JMP   (796)     =62097: CLOSE:  File (Nummer im Accu) schliessen
65478 JMP   (798)     =61966: CHKIN:  Fileeingabevorbereitung
65481 JMP   (800)     =62032: CHKOUT: Fileausgabevorbereitung
65484 JMP   (802)     =62259: CLRCHN: aktive I/O-Kanaele schliessen
65487 JMP   (804)     =61783: CHRIN:  Zeichen vom aktiven I-Kanal in Accu
65490 JMP   (806)     =61898: CHROUT: Ausgabe Accu auf aktiven O-Kanal
65493 JMP  62622      LOAD:   Load und Verify von Programmen
65496 JMP  62941      SAVE:   Speichern von Programmen
65499 JMP  63204      SETTIM: Uhrzeit setzen (Accu, XR, YR)
65502 JMP  63197      RDTIM:  Uhrzeit in 60stel Sekunden in Accu, XR, YR
65505 JMP   (808)     =63213: STOP:   Abfrage STOP-Taste
65508 JMP   (810)     =61758: GETIN:  Zeichen von aktivem Eingabekanal holen
65511 JMP   (812)     =62255: CLALL:  alle Files schliessen
65514 JMP  63131      UDTIM:  Uhr um eine 60stel Sekunde weitersetzen
65517 JMP  58629      SCREEN: Lesen des Bildschirmformats (X/Y)
65520 JMP  58634      PLOT:   Lesen und Setzen der Cursorposition (X/Y)
65523 JMP  58624      IOBASE: Startadresse der IRQ-CIA nach (XR/YR)

65526  82  82  66  89
```

Hardware-Interrupt-Vektoren

```
65530  67 254   65091   NMI
65532 226 252   64738   RESET
65534  72 255   65352   IRQ
```

Adressumrechnung (Betriebssystemlisting) fuer den VIC-20

Das Betriebssystem des Commodore 64, das von der Struktur her mit dem des VIC-20 ziemlich identisch ist, liegt in zwei getrennten Bloecken zu je acht Kilobytes im Adressbereich des Computers. Dies sind die Bereiche von 40960 bis 49151 sowie 57344 bis 65535. Beim VIC-20 belegt es einen durchgehenden Block von 16 KB im Bereich von 49152 bis 65535. Bei beiden Geraeten laesst sich das Betriebssystem in zwei Bereiche aufteilen: BASIC und KERNAL. BASIC (der erste Bereich von acht KB) enthaelt den BASIC-Interpreter, wohingegen KERNAL Routinen zur Abwicklung von I/O und anderen hardwarespezifischen Dingen enthaelt.

Aufgrund der aehnlichen Strukturierung der beiden Betriebssysteme (womit ein gleichartiger Aufbau gemeint ist, denn allein durch die unterschiedliche Lage der beiden Betriebssysteme ist ein gleicher Aufbau nicht moeglich) ist es, im Gegensatz zu Betriebssystemen von anderen Commodore-Computern (wenn man von den CBM 4001 und CBM 8001-Serien absieht, da deren Betriebssystem bis auf einen Bereich von 2 Kilobytes absolut identisch ist) moeglich, fuer den groessten Teil der Routinen des Commodore 64 den entsprechenden Bereich des VIC-20 anzugeben.

Im folgenden nun eine Gegenueberstellung der Betriebssystembereiche des Commodore 64 zu denen des VIC-20. Bei den von der Struktur her gleichen Teilen wurde der Wert angegeben, der den Wert des Commodore 64 in den fuer den VIC-20 richtigen Wert umwandelt. War die Struktur zu unterschiedlich, so ist kein entsprechender Wert angegeben. Solche Programmsegmente werden jedoch auch nur aeussert selten aufgerufen. Manche Stellen des einen Betriebssystems existieren im anderen nicht. Auch werden manche Bereiche wohl in kaum einem Programm aufgerufen, sodass nur ein Teil der Uebersicht praktisch genutzt werden kann, speziell beim Umschreiben von Programmen:

Commodore 64		VIC-20		
40960 - 49148		49152 - 57340		+8192
49149 - 49151	JMP	-------------		
57344 - 57732		57341 - 57729		-3
57733 - 57734	BNE	57730 - 57731	BEQ	
-------------		57732 - 57734	JMP	
57735 - 57783		57735 - 57783		0
57784 - 57789		57784 - 57786		
57790 - 58234		57787 - 58231		-3
58235 - 58245		58471 - 58481		+236
58246 - 58250	LDX#, JMP	58482 - 58484	JMP	
58251 - 58259	Korrektur	-------------		
58260 - 58271		58232 - 58243		-28
58272 - 58273	BNE	58244 - 58246	JMP	
58274 - 58306		58247 - 58279		-27

Umlegung des USR-Vektors von (0,1,2) nach (784,785,786)

58307 - 58309	STA 784	58280 - 58281	STA 0	
58310 - 58313		58282 - 58285		-28
58314 - 58319	STA 785 STY 786	58286 - 58289	STA 1 STA 2	
58320 - 58438		58290 - 58408		-30
58439 - 58462		58447 - 58470		+8
58463 - 58540	RESET-Meldung	58409 - 58446		
58541 - 58550	Korr. CHKOUT	-------------		
58551 - 58585	Fuellcodes	58492 - 58527		

Commodore 64		VIC-20		
58586 - 58591	Zusatz (Screen)	-------------		
58592 - 58603	Test CBM-Key	-------------		
58604 - 58623	RS-232 PAL	-------------		
58624 - 58650		58624 - 58650		0

Startadresse des Bildschirms in VIC-Chip bringen

Commodore 64		VIC-20		
-------------		58651 - 58677		
58651 - 58919		58678 - 58946		+27
-------------		58947 - 58948	???	
58920 - 58941		58949 - 58970		+29
-------------		58971 - 58993	NOPs	
58942 - 59101		58994 - 59153		+52
59102 -		59154 - 59156		
- 59112	sowie	60763 - 60776	Korrektur	
59113 - 59347		59157 - 59391		+44
-------------		59392 - 59412	NOPs	
59348 - 59355		59413 - 59420		+65
-------------		59421 - 59426	NOPs	
59356 - 59609		59427 - 59680		+71
59610 - 59625	16 Farbcodes	59681 - 59688	8 Farbcodes	
-------------		59689 - 59764	???	
59626 - 59643		59765 - 59782		+139

Umlegung der Adresse (242) nach (677)

Commodore 64		VIC-20		
59644 - 59646	DEC 677	59783 - 59784	DEC 242	
59647 - 59700		59785 - 59838		+138
59701 - 59703	INC 677	59839 - 59840	INC 242	
59704 - 59755		59841 - 59892		+137
59756 - 59758	STX 677	59893 - 59894	STX 242	
59759 - 59767		59895 - 59903		+136
59768 - 59770	LDX 677	59904 - 59905	LDX 242	
59771 - 59794		59906 - 59929		+135
59795 - 59797	CPX 677	59930 - 59931	CPX 242	
59798 - 59818		59932 - 59952		+134
59819 - 59821	CPX 677	59953 - 59954	CPX 242	
59822 - 59838		59955 - 59971		+133
59839 - 59841	LDX 677	59972 - 59973	LDX 242	
59842 - 59844		59974 - 59976		+132

"direkt" steht fuer "direkte Ausfuehrung statt Subroutine"

Commodore 64		VIC-20		
59845 - 59847	JMP	59977 - 59989	direkt	
59848 - 60000		59990 - 60142		+142

Bedienung des Prozessorports (PP) durch Zeropage

Commodore 64		VIC-20		
60001 - 60002	LDA 1	60143 - 60145	LDA 37151	
60003 - 60010		60146 - 60153		+143
60011 - 60012	LDA 1	60154 - 60156	LDA 37148	
60013 - 60020		60157 - 60164		+144
60021 - 60022	LDA 1	60165 - 60167	LDA 37148	
60023 - 60024		60168 - 60169		+145
-------------		60170 - 60174		
60025 - 60026	STA 1	60175 - 60177	STA 37148	
60027 - 60057		60178 - 60208		+151
60058		60214 - 60215		
60059 - 60066		60216 - 60223		+157
60067 - 60071		60209 - 60213		+142
60072 - 60073		60224 - 60225		+152
60074	PHA	-------------		
60075 - 60116		60226 - 60267		+151
60117 - 60121		60268 - 60270		

Commodore 64		VIC-20		
60122 - 60123		60271 - 60272		+149
60124	PLA	-------------		
60125 - 60248		60273 - 60396		+148
-------------		60397 - 60415	NOPs	
60249 - 60256		60416 - 60423		+167
-------------		60424 - 60427	NOPs	
60257 - 60264		60428 - 60435		+171
-------------		60436 - 60471	NOPs	
60265 - 60266	LDA #6	-------------		
60267 - 60280		60472 - 60485		+205
60281 - 60288	Decode/Pointer	60486 - 60509		
60289 - 60483		60510 - 60704		+221
60484 - 60535		60705 - 60762		
60536 - 60600		60777 - 60841		+241
-------------		60842 - 60899	???	
60601 - 60645	Data fuer VIC	60900 - 60915		
60646 - 60655		60916 - 60925		+270
60656 - 60680	LSBs (Screen)	60926 - 60947		
60681 - 60706		60948 - 60973		+267
60707	SEI	-------------		
60708 - 60725		60974 - 60991		+266
60726	SEI	-------------		
60727 - 60742		60992 - 61007		+265
-------------		61008	LSR	
60743 - 60754		61009 - 61020		+266
-------------		61021	LSR	
60755 - 60759		61022 - 61026		+267
-------------		61027	LSR	
60760 - 60764		61028 - 61032		+268
-------------		61033	LSR	
60765 - 60781		61034 - 61050		+269
60782	ASL	61051 - 61052	LSR, LSR	
60783 - 60822		61053 - 61092		+270
60823 - 60830	Start Timer	-------------		
60831 - 60840		61093 - 61102		+262
-------------		61103	LSR	
60841 - 60910		61104 - 61173		+263
60911	SEI	-------------		
60912 - 60964		61174 - 61229		+262
60965 - 60975	Start Timer	-------------		
60976 - 60985		61230 - 61239		+254
60986 - 60989	BMI, BPL	61240 - 61243	BCS, BCC	+254
60990 - 61025		61244 - 61279		+254
61026 - 61028	ASL, BPL	61280 - 61282	LSR, BCC	+254
-------------		61283	LSR	
61029 - 61038		61284 - 61293		+255
61039 - 61041	ASL, BMI	61294 - 61296	LSR, BCS	+255
61042 - 61048		61297 - 61303		+255
61049 - 61052	BIT, BVC	61304 - 61307	LDA, BEQ	+255
61053 - 61078		61308 - 61333		+255
61079 - 61106		58528 - 58555		-2551
61107 - 61114	Zaehlschleife	61334 - 61346	Timer	
61115 - 61240		61347 - 61472		+232
61241 - 61257	NMI sperren	61473 - 61478		
61258 - 61314		61479 - 61535		+221
61315 - 61320	NMI-Register	-------------		
61321 - 61324		61536 - 61539		+215
61325 - 61327	JMP	61540 - 61543	STA, RTS	
61328 - 61408		61544 - 61624		+216
-------------		61625 - 61627	ILL DEV #	
61409 - 61452		61628 - 61671		+219
61453 - 61457	direkt	61672 - 61674	JSR	
61458 - 61459		61675 - 61676		+217
61460 - 61462	JSR	-------------		

Commodore 64		VIC-20		
61463 - 61475		61677 - 61689		+214
61476 - 61477	LDA 158	-------------		
61478 - 61479		61690 - 61691		+212
61480 - 61490	Timer	61692 - 61697		
61491 - 61502		61698 - 61709		+207
61503 - 61516		61710 - 61717		
61517 - 61537		61718 - 61738		+201
61538 - 61543		61739 - 61743		
61544 - 61558		61744 - 61758		+200
61559 - 61564		61759 - 61765		
61565 - 61573		61766 - 61774		+201

Belegung von Bit 3 im RS-232 Status beim Commodore 64

Commodore 64		VIC-20		
61574 - 61576	LDA	-------------		
61577 - 61584		61775 - 61782		+198
61585 - 61589	AND#, STA	-------------		
61590 - 61595		61783 - 61788		+193
61596 - 61600	ORA#, STA	-------------		
61601 - 61621		61789 - 61809		+188
61622 - 61626	LDA#, STA	-------------		
61627 - 61886		61810 - 62069		+183
61877 - 61897		62070 - 62073		
61898 - 61914		62074 - 62090		+176
61915	LSR	62091 - 62094	CMP#, BEQ	
61916 - 61918		62095 - 62097		+179
-------------		62098	PHA	
61919 - 61922		62099 - 62102		+180
61923 - 61924	BCC	-------------		
61925 - 61952		62103 - 62130		+178
61953 - 61954	LDA 158	62131	PLA	
61955 - 61959		62132 - 62136		+177
-------------		62137 - 62141		
61960 - 61962		62142 - 62144		+182
61963 - 61965	JMP	62145 - 62150	direkt	
61966 - 62126		62151 - 62311		+185
62127 - 62129	JSR	62312 - 62326	direkt	
62130 - 62162		62327 - 62359		+197
62163	SEC	-------------		
62164 -		62360 - 62367		
- 62175	sowie	58575 - 58585	Korrektur	
62176 - 62428		62368 - 62620		+192
62429 - 62432	CLEAR ST	-------------		
62433 - 62469		62621 - 62657		+188
62470 - 62472	JMP	62658 - 62662	direkt	
62473 - 62475	JSR	62663 - 62677	direkt	
62476 - 62503		62678 - 62705		+202
62504 - 62540	sowie	62706 -		
65326 - 65344	Korrektur	- 62731		
62541 - 62594		62732 - 62785		+191
62595 - 62621	Init RS-232	-------------		
62622 - 62654		62786 - 62818		+164
62655 -		62819 - 62821		
- 62659	sowie	58556 - 58560	Korrektur	
62660 - 62692		62822 - 62854		+162
62693 -		62855 - 62857		
- 62706	sowie	58561 - 58574	Korrektur	
62707 - 62770		62858 - 62921		+151
62771 - 62773	LSR, BCS	62922 - 62925	CMP#, BNE	
62774 - 63064		62926 - 63216		+152
63065 - 63067	LSR, BCS	63217 - 63220	CMP#, BNE	
63068 - 63163		63221 - 63316		+153
63164 - 63196	Update RUNSTOP	63317 - 63327		
63197 - 63328		63328 - 63459		+131

Commodore 64		VIC-20		
63329 - 63334	Test CBM-Key	-------------		
63335 - 63533		63460 - 63658		+125
63534 - 63543	PP	63659 - 63670		
63544 - 63612		63671 - 63739		+127
63613 - 63625	Timer Init	-------------		
63626 - 63628		63740 - 63742		+114
63629 - 63636	BLANK SCREEN	-------------		
63637 - 63658		63743 - 63764		+106
63659 - 63664	PP	63765 - 63774		
63665 - 63676		63775 - 63786		+110
-------------		63787 - 63789	STA	
63677 - 63689		63790 - 63802		+113
-------------		63803 - 63812	Update Time	
63690 - 63760		63813 - 63883		+123
63761 - 63785	Zusatz	-------------		
63786 - 63787		63884 - 63885		+98
63788 - 63810		63886 - 63908		+98
63811 - 63821	Zusatz	-------------		
63822 - 63837		63909 - 63924		+87
-------------		63925 - 63927	CLEAR IFR	
63838 - 63919		63928 - 64009		+90
63920 - 63931		64010 - 64014		
63932 - 63935		64015 - 64018		+83
63936 - 63938	Zusatz	-------------		
63939 - 64025		64019 - 64105		+80
64026 - 64030	BMI, JMP	64106 - 64107	BPL	
64031 - 64331		64108 - 64408		+77
64332 - 64340	Zusatz	-------------		
64341 - 64438		64409 - 64506		+68
64439 - 64446	Start Timer	-------------		
64447 - 64452	PP	64507 - 64514		
64453 - 64654		64515 - 61716		+62
64655 - 64656	SAVE >32767	-------------		
64657 - 64660		64717 - 64720		+60
64661 - 64668	UNBLANK SCREEN	-------------		
64669 - 64676		64721 - 64728		+52
64677 - 64679		64729 - 64741		
64680 - 64713		64742 - 64775		+62
64714 - 64720	PP	64776 - 64784		
64721 - 64750		64785 - 64814		+64
64751 - 64753	RESET VIC-II	-------------		
64754 - 64850		64815 - 64911		+61
64851 - 64853	Absolute	64912 - 64913	Zeropage	
64854 - 64870		64914 - 64930		+60
-------------		64931 - 64937		
64871 -	RAM-Test &c.	64938 - 65008		
- 64922	sowie	65169 - 65192		
64923 - 64930		65009 - 65016		+86
64931 - 65016	I/O-Reset	65017 - 65096		
65017 - 65039		65097 - 65119		+80
65040	PHA	-------------		
65041 - 65045		65120 - 65124		+79
65046	PLA	-------------		
65047 - 65090		65125 - 65168		
65091 - 65099		65193 - 65201		+102
65100 - 65109	Test RESTORE	65202 - 65214		
65110 - 65117		65215 - 65222		+105
-------------		65223 - 65225	CLEAR IFR	
65118 - 65137		65226 - 65245		+108
65138 - 65217	sowie	65246 -		
65238 - 65325		- 65371		
65218 - 65237		65372 - 65393		
65345 - 65351	Zusatz (Tape)	-------------		
65352 - 65370		65394 - 65412		+42

Commodore 64		VIC-20	
65371 - 65407	PAL/NTSC-Test	-------------	
65408	Fuellcodes	65413 - 65417	
65409 - 65417	Vektoren	-------------	
65418 - 65525		65418 - 65525	0
65526 - 65529	Fuellcodes	65526 - 65529	
65530 - 65535		65530 - 65535	0

SYSTEMROUTINEN

Das Betriebssystem des Commodore 64 enthaelt im Bereich von 65409 bis 65525 eine Sprungtabelle, anhand derer elementare Routinen fuer eigene Programme angesprochen werden koennen. Diese Sprungtabelle wird auch vom Betriebssystem selbst genutzt, wodurch es einfach wird, neue Funktionen zu implementieren, da einige JMP's der Tabelle indirekte Spruenge ueber Vektoren sind.

Diese Sprungtabelle ist, von den ersten drei Eintragungen abgesehen, auch im VIC-20 implementiert, so dass Routinenaufrufe in Maschinenprogrammen, die fuer den VIC-20 geschrieben sind, im Normalfall nicht geaendert werden muessen, falls sie ueber die Sprungtabelle ablaufen.

Auch die anderen Serien von Commodore (PET, CBMs) haben eine Sprungtabelle am Ende des Betriebssystems, die jedoch nur zum Teil mit der des Commodore 64 identisch ist, so dass eine Programmaenderung wohl meist unerlaesslich sein wird.

Handhabung der Systemroutinen:

Die Systemroutinen sind so ausgelegt, dass nach Aufruf der Routine Fehler abgefangen werden koennen (sofern ueberhaupt welche auftreten koennen). Es kommt daher nicht zum Abbruch des Programms durch eine Fehlermeldung, sondern es erfolgt eine Rueckkehr zur aufrufenden Routine. Das Auftreten eines Fehlers wird durch eine Rueckkehr mit gesetztem Carry gekennzeichnet. In diesem Fall enthaelt der Accu die Nummer des Fehlers, der dann durch das Programm behandelt werden kann.

Einige Routinen benoetigen auch Informationen, die durch andere Routinenaufrufe gegeben worden sein muessen (Vorbereitungsroutinen), da sonst die Moeglichkeit besteht, dass die Routine nicht erwartungsgemaess funktioniert.

Bedeutungen der Fehlernummern:

Fehlernummer bei gesetztem Carry im Accu:

0: BREAK
   gedrueckte RUNSTOP-Taste waehrend des Programmablaufs
1: TOO MANY FILES
   die maximale Anzahl an offenen Files betraegt zehn
2: FILE OPEN
   jeder Fileeintrag muss eine andere Filenummer haben
3: FILE NOT OPEN
   jedes File muss vor Zugriff geoeffnet werden
4: FILE NOT FOUND
   das gesuchte File ist nicht verfuegbar
5: DEVICE NOT PRESENT
   das angesprochene Geraet reagiert nicht auf Adressierung
6: NOT INPUT FILE
   aus Schreibfiles kann nicht gelesen werden
7: NOT OUTPUT FILE
   in Lesefiles kann nicht geschrieben werden
8: MISSING FILE NAME
   bei LOAD und SAVE (serieller Bus) ist ein Filename noetig
9: ILLEGAL DEVICE NUMBER
   versuchtes Kommando ist bei diesem Geraet nicht moeglich

Eine Fehlermeldung selbst wird nicht ausgegeben; dies muss, sofern erwuenscht, vom Hauptprogramm aus durchgefuehrt werden. Es ist jedoch durch den Aufruf der Routine SETMSG (siehe Beschreibung) moeglich, die Ausgabe einer Fehlermeldung zu erreichen, die dann lautet "I/O.ERROR #?", die jedoch zu keinem Programmabbruch fuehrt. Die Fehlerausgaberoutine beginnt ab Adresse 63227, hat jedoch mehrere Einsprungstellen.

Routinenbeschreibung:

65409: Feststellung der Fernsehnorm (PAL/NTSC)

Aufgrund der unterschiedlichen Fernsehnormen besitzt der Commodore 64 je nach Fabrikation einen Quarz entweder fuer PAL oder NTSC. Aus der Frequenz dieses Quarzes werden dann andere benoetigte Frequenzen generiert. So auch die normale Taktfrequenz von ungefaehr einem MHz.

Der Quarz hat folgenden Wert: PAL: 17.734472 MHz
NTSC: 14.31818 MHz

Die Taktfrequenz fuer die CPU und alle anderen davon abhaengigen Bausteine errechnet sich aus der Quarzfrequenz durch Division durch 18 (PAL) beziehungsweise 14 (NTSC). Daraus ergibt sich, dass der europaeische Commodore 64 etwas langsamer als der amerikanische ist. Im normalen BASIC sind diese Unterschiede unbedeutend, bei Datenuebertragungen der RS-232-Schnittstelle muessen jedoch diese unterschiedlichen Taktfrequenzen beachtet werden. Auch die Initialisierung fuer den Interrupt muss diesen Unterschied beachten, da die interne Uhr genau alle 60stel Sekunden weitergestellt werden muss.

Diese Routine setzt in Adresse 678 eine von der Frequenz abhaengige Flag und setzt automatisch auch den Timer fuer den Interrupt zur Tastaturabfrage neu fest. Ausserdem wird ein Screen-Editor-Reset ausgefuehrt.

Die Adresse 678 hat folgenden Inhalt: PAL: 1
NTSC: 0

65412: I/O-RESET

Beide CIAs sowie der interne I/O-Port der 6510 werden initialisiert. Ausserdem wird der TimerA der IRQ-CIA mit dem fuer den normalen Tastaturabfrage-Interrupt notwendigen Wert initialisiert (siehe 65409).

65415: Pruefung auf freien BASIC-RAM-Bereich

Diese Routine prueft, bis zu welcher Adresse fuer BASIC verwendbares RAM vorhanden ist und setzt die Pointer fuer MEMBOT und MEMTOP

65418: RESTOR

Ruecksetzung der Sprungvektoren im Bereich von 788 bis 819 auf die Normalwerte.

65421: VECTOR

Parameter: XR, YR, Carry

Bei gesetztem Carry wird der Sprungvektorbereich (788,...,819) in den Bereich uebertragen, auf den das Registerpaar (XR/YR) zeigt. Dort kann die Sprungtabelle dann modifiziert werden. Das Rueckschreiben in den Bereich der Vektortabelle erfolgt mit geloeschtem Carry und der Startadresse der modifizierten Vektortabelle in (XR/YR).

65424: SETMSG

Parameter: Accu

Die hauptsaechliche Routine besteht im Prinzip fast nur aus dem Befehl 'STA 157'. Doch sollte normalerweise diese Routine aufgerufen werden, da die Adresse 157 in anderen Versionen des Betriebssystems eine andere Aufgabe haben koennte. Die Adresse 157 hat zwei Bedeutungen, die in den Bits 6 und 7 festgelegt werden. Bit 7 gibt an, ob Meldungen wie "SEARCHING" ausgegeben werden sollen. Ist Bit 7 gesetzt (ist im Direktmodus der Fall), so werden die Meldungen ausgegeben, ansonsten unterdrueckt. Ist Bit 6 gesetzt, so werden auch die I/O-Errors ausgegeben. Wird eine Routine angesprungen, bei der Fehlermoeglichkeiten bestehen, so wird, falls ein Fehler auftritt, die Meldung "I/O ERROR #?" mit der entsprechenden Nummer ausgegeben, das Programm jedoch nicht abgebrochen. Der Accu enthaelt nach Aufruf der Routine den Wert der Statusvariablen fuer Recorder und seriellen Bus.

65427: SECOND

Parameter: Accu

Vorbereitungsroutinen: LISTEN

moegliche Fehler: siehe READST (65463)

Ausgabe der Sekundaeradresse (im Accu) auf den seriellen Bus nach LISTEN. Fuer die Ausgabe der Sekundaeradresse nach TALK kann diese Routine nicht benutzt werden.

65430: TKSA

Parameter: Accu

Vorbereitungsroutinen: TALK

moegliche Fehler: siehe READST (65463)

Ausgabe der Sekundaeradresse (im Accu) auf den seriellen Bus nach TALK. Fuer die Ausgabe der Sekundaeradresse nach LISTEN kann diese Routine nicht benutzt werden.

65433: MEMTOP

Parameter: XR, YR, Carry

Bei gesetztem Carry wird die Adresse der hoechsten verfuegbaren RAM-Adresse in das Registerpaar (XR/YR) uebertragen, bei geloeschtem Carry wird (XR/YR) in den Pointer auf die hoechste RAM-Adresse geschrieben.

65436: MEMBOT

Parameter: XR, YR, Carry

Uebergabe der BASIC-RAM-Startadresse (im Normalfall 2048) in (XR/YR) bei gesetztem Carry und Uebertragung von (XR/YR) in den Pointer auf den Anfang des BASIC-RAMs bei geloeschtem Carry.

65439: SCNKEY

Abfrage der Tastatur wie durch das normale Interrupt-Handling auch. Bei gedrueckter Taste wird der ASCII der Taste im Tastaturpuffer abgelegt. Der Aufruf dieser Routine ist notwendig, falls bei gesperrtem IRQ trotzdem Tastendruecke erkannt werden sollen.

65442: SETTMO

Parameter: Accu

Diese Routine dient zur Handhabung des Timeouts fuer den seriellen Bus und setzt eine Flag, die im Falle eines Timeouts abgefragt wird, so dass die Moeglichkeit besteht, einen weiteren Versuch zu machen, auf den seriellen Bus zuzugreifen. Allerdings wird die Flag im gesamten Betriebssystem nicht genutzt, so dass diese Routine keinen Sinn hat. Es besteht also keine Moeglichkeit (ausser, wenn dies durch das eigene Programm realisiert wird), den Computer ein Timeout uebersehen zu lassen (im Gegensatz zu den CBMs).

65445: ACPTR

Parameter: Accu

Vorbereitungsroutinen: TALK (, TKSA)

moegliche Fehler: siehe READST (65463)

Diese Routine holt ein Byte vom seriellen Bus in den Accu. Das Geraet muss zuvor mittels TALK (und evtl. TKSA) angesprochen worden sein.

65448: CIOUT

Parameter: Accu

Vorbereitungsroutinen: LISTEN (, SECOND)

moegliche Fehler: siehe READST (65463)

Diese Routine gibt den Inhalt des Accumulators auf den seriellen Bus aus Wird das Byte nicht empfangen, so wird ein Timeout gegeben. Ein Byte wird durch diese Routine immer in einem Zwischenpuffer (149) gehalten, um im Falle eines UNLSNs dieses Byte zusammen mit der EOI-Kennzeichnung ausgeben zu koennen.

65451: UNTLK

moegliche Fehler: siehe READST (65463)

Senden eines Untalks ueber den seriellen Bus an den momentanen Talker.

65454: UNLSN

moegliche Fehler: siehe READST (65463)

Senden eines Unlistens ueber den seriellen Bus an den momentanen Listener.

65457: LISTEN

Parameter: Accu

moegliche Fehler: siehe READST (65463)

An das Geraet, dessen Geraetenummer sich im Accu befindet, wird ein LISTEN ausgegeben. Attention bleibt nach Rueckkehr aktiv und wird erst nach Ausgabe der Sekundaeradresse (SECOND) rueckgesetzt.

65460: TALK

Parameter: Accu

moegliche Fehler: siehe READST (65463)

An das Geraet, dessen Geraetenummer sich im Accu befindet, wird ein TALK ausgegeben. Attention bleibt nach Rueckkehr aktiv und wird erst nach Ausgabe der Sekundaeradresse (TKSA) rueckgesetzt.

65463: READST

Parameter: Accu

Nach Aufruf dieser Routine enthaelt der Accu den momentanen Wert der Statusvariablen. Ist der Inhalt des Registers fuer die Geraetenummer (186) gleich zwei, so wird der RS-232-Status in den Accu uebertragen, wobei das RS-232-Statusbyte danach durch Belegen mit dem Wert null geloescht wird. Fuer eine zweite Abfrage desselben Werts kann also nicht diese Routine aufgerufen werden. Die Bedeutung des RS-232-Statusworts ist der Erklaerung des RS-232-Handlings zu entnehmen. Der Aufruf dieser Routine erfolgt normalerweise nach Eroeffnung eines neuen Kanals.

Die Bedeutungen der Statusbits (Bus und Tape)

ST Bit	Bit-Wert	Recorder (nur Read)	serieller Bus (Read und Write)
0	1	-	Zeitfehler, Wrt
1	2	-	Zeitfehler, Rd, Wrt
2	4	zu kurzer Block	-
3	8	zu langer Block	-
4	16	Lesefehler	-
5	32	Pruefsummenfehler	-
6	64	Fileende (EOF)	Uebertragungsende/EOI
7	128	-	Geraet reagiert nicht

EOF tritt nur im Zusammenhang mit Files auf. Bei VERIFY zeigt Bit 4 ausserdem Nichtuebereinstimmungen an.

65466: SETLFS

Parameter: Accu, XR, YR

Festsetzung der Filenummer (Accu), Geraetenummer (XR) und der Sekundaeradresse (YR). Diese Routine muss vor Eroeffnung eines Files aufgerufen werden. Die Geraetenummern haben folgende Bedeutungen:

0: Tastatur
1: Recorder
2: RS-232-Kanal
3: Bildschirm

Geraetenummern ab 4 sprechen den seriellen Bus an. Dabei ist die Nummer vier meist dem Drucker vorbehalten wohingegen die Nummer acht im Normalfall fuer Diskettenstationen vorgesehen ist.

65469: SETNAM

Parameter: Accu, XR, YR

Diese Routine legt die Daten fuer den Filenamen eines Files fest und muss vor Eroeffnung eines Files aufgerufen werden. Ist die Angabe eines Filenamens nicht erwuenscht, so muss als Laenge des Filenamens der Wert null verwendet werden. Die Laenge des Filenamens wird im Accu festgelegt, die Startadresse wird durch das Registerpaar (XR/YR) festgelegt (niederwertiges Byte im XR, hoeherwertiges Byte im YR).

65472: OPEN

Vorbereitungsroutinen: SETLFS, SETNAM

moegliche Fehler: 0 (nur Tape), 1, 2, 4, 5, 6
(falls File# = 0)

Der Aufruf dieser Routine bewirkt das Eroeffnen eines logischen Files mit den durch SETLFS und SETNAM gegebenen Parametern und die Eintragung dieses Files in die Tabelle der Fileparameter.

65475: CLOSE

Parameter: Accu

moegliche Fehler: 0 (Tape Write)

Ordnungsgemaesses Schliessen des Files mit der im Accu befindlichen Nummer und Loeschen des Eintrags in der Filetabelle.

65478: CHKIN

Parameter: XR

Vorbereitungsroutinen: OPEN

moegliche Fehler: 0 (Tape Read), 3, 5, 6

Vor Zugriff auf ein durch OPEN eroeffnetes File durch GETIN oder CHRIN (eine Ausnahme bildet die Tastatur, die nicht durch CHKIN aktiviert werden muss) muss diese Routine aufgerufen werden. Sie aktiviert das mit der Filenummer (im XR) verbundene Geraet zum Zugriff in dieses File. Ist die Geraetenummer groesser als 3, so wird ein TALK mit anschliessender Ausgabe der Sekundaeradresse durchgefuehrt.

65481: CHKOUT

Parameter: XR

Vorbereitungsroutinen: OPEN

moegliche Fehler: 0 (Tape Write), 3, 5, 7

Sollen Daten auf ein zuvor durch OPEN eroeffnetes File durch CHROUT ausgegeben werden (Ausgaben auf den Bildschirm koennen direkt erfolgen), so ist der Aufruf dieser Routine notwendig. Sie bereitet das Geraet, dessen Filenummer im XR angegeben wird, auf Ausgaben in dieses File vor. Ist die Geraetenummer groesser als 3, so wird ein LISTEN mit anschliessender Ausgabe der Sekundaeradresse durchgefuehrt.

65484: CLRCHN

Nach Ausgabe oder Empfang von Daten auf oder von Files wird diese Routine benutzt, um alle offenen Kanaele zu schliessen (UNTLK und UNLSN) und die Standardwerte zu setzen (Eingabe von Tastatur, Ausgabe auf Bildschirm). Wird diese Routine nach Aktivierung nicht aufgerufen, so koennen mehrere Geraete aktiv am I/O-Bus bleiben und so zum Beispiel die Ausgabe von Diskettendaten direkt auf Drucker erfolgen.

65487: CHRIN

Parameter: Accu

Der Aufruf dieser Routine bewirkt das Holen eines Zeichen aus dem durch CHKIN aktivierten File oder, falls kein CHKIN erfolgt ist, von der Tastatur. Das geholte Zeichen befindet sich im Accu. Der Kanal zu diesem File bleibt geoeffnet. Sollen Daten von der Tastatur geholt werden, so wird der Cursor eingeschaltet und so lange gewartet, bis die RETURN-Taste gedrueckt wird. Bei jedem Aufruf dieser Routine wird dann jeweils ein Zeichen geholt. Ist das letzte Zeichen gelesen worden, so wird der Code 13 geschickt. Bei weiteren Aufrufen beginnt der Vorgang von vorne.

65490: CHROUT

Parameter: Accu

Ausgabe eines Zeichens (im Accu) auf den aktiven Ausgabekanal. Wird durch CHKOUT kein Ausgabekanal festgelegt, so erfolgt die Ausgabe auf den Bildschirm. Der Kanal bleibt nach Ausgabe des Zeichens geoeffnet.

65493: LOAD

Parameter: Accu, XR, YR

Vorbereitungsroutinen: SETLFS, SETNAM

moegliche Fehler: 0, 4, 5, 8 (nur Bus), 9

Der Accu enthaelt die Flag fuer LOAD (0) und VERIFY (1). (XR/YR) gibt die Startadresse an, ab der das Programm abgelegt beziehungsweise verglichen werden soll (nur fuer Sekundaeradresse ungleich null). Wird als Sekundaeradresse null spezifiziert, so wird das Programm an die Stelle geladen, ab der es gespeichert worden ist. Bei der Rueckkehr enthaelt (XR/YR) die Endadresse plus eins des geladenen oder verglichenen Programms. Ein Filename muss beim Laden von Recorder nicht spezifiziert werden.

65496: SAVE

Parameter: Accu, XR, YR

Vorbereitungsroutinen: SETLFS, SETNAM

moegliche Fehler: 0, 5, 8 (nur Bus), 9

Der Accu gibt den Pointer auf die Startadresse des Programms an. Soll ab Anfang des BASIC-Bereichs gespeichert werden, so muss der Accu den Wert 43 enthalten, da der Pointer (43/44) auf den Anfang des BASIC-Programms zeigt. Das Registerpaar (XR/YR) enthaelt die Endadresse des zu speichernden Programms. Die Angabe eines Filenamens ist bei Ausgabe auf Recorder nicht notwendig.

65499: SETTIM

Parameter: Accu, XR, YR

Diese Routine setzt die interne Uhr auf den in den Registern festgesetzten Wert neu fest. Der Wert muss in 60stel Sekunden angegeben werden, wobei der Accu das hoechstwertigste und das YR das niederwertigste Byte enthaelt.

65502: RDTIM

Parameter: Accu, XR, YR

Das Lesen der internen Uhr erfolgt durch Aufruf dieser Routine. Die Register enthalten dann die Uhrzeit in 60stel Sekunden, wobei der Accu das hoechstwertigste und das YR das niederwertigste Byte enthaelt.

65505: STOP

Parameter: Zeroflag

Ist bei Aufruf dieser Routine die RUNSTOP-Taste gedrueckt, so wird die Zeroflag gesetzt. Der Zustand der uebrigen Flags bleibt unveraendert. Bei gedrueckter RUNSTOP-Taste wird ausserdem ein CLRCHN durchgefuehrt.

65508: GETIN

Parameter: Accu

GETIN ist identisch mit CHRIN, mit dem Unterschied, dass, wird ein Zeichen von der Tastatur gelesen, direkt aus dem Tastaturpuffer gelesen und nicht auf die Eingabe eines Zeichen gewartet wird. Ist der Tastaturpuffer leer, so wird dem Accu der Wert null uebergeben. Der Tastaturpuffer wird durch SCNKEY gefuellt, was normalerweise durch die normale Interruptroutine geschieht.

65511: CLALL

Diese Routine setzt den Zaehler fuer die Anzahl offener Files auf null zurueck und fuehrt anschliessend ein CLRCHN aus. Es erfolgt kein ordnungsgemaesses Schliessen noch offener Dateien!

65514: UDTIM

Im Normalfall wird diese Routine von der normalen Interruptroutine aufgerufen. Sie erhoeht die interne Uhr um eine 60stel Sekunde. Ausserdem wird das Register fuer die RUNSTOP-Abfrage (Flag fuer diverse Tasten) auf den aktuellen Stand gebracht, um die RUNSTOP-Funktion aufrecht zu erhalten.

65517: SCREEN

Parameter: XR, YR

Nach Aufruf dieser Routine enthaelt das XR die Anzahl Spalten (40) und das YR die Anzahl Zeilen (25) des Bildschirms.

65520: PLOT

Parameter: XR, YR, Carry

Bei gesetztem Carry enthaelt das XR die aktuelle Cursorzeile (0 bis 24) sowie das YR die aktuelle Cursorspalte (0 bis 79). Bei geloeschtem Carry werden die Cursorzeile und -spalte entsprechend den Inhalten von XR und YR gesetzt. Ausserdem werden die zugehoerigen Zeilenparameter aktualisiert.

65523: IOBASE

Parameter: XR, YR

Diese Routine uebergibt in (XR/YR) die Startadresse des I/O-Bereichs, beim Commodore 64 ist dies die Startadresse der IRQ-CIA. Die Registerfolge und die Registerbedeutung haengt vom I/O-Baustein ab und ist daher beim VIC-20 und C=64 unterschiedlich.

Der neue Basicstandard
für Commodore Computer ˣ

ˣ CBM + VC ›VolksComputer‹ + Commodore 64

Hilfsfunktionen:
FIND Suchen nach Text, Variablen, mit Bereichsangabe. • **AUTO** automatische Zeilennummerngenerierung. • **DEL** Zeilennummernbereich löschen. • **RENUM** Programmzeilen Umnumerierung. • **TRACE (OFF)** Einzelschritt- oder langsamer Programmablauf mit Einblendung der jeweils abgearbeiteten Zeile (vollständige Zeile). • **ON/OFF** Ein-/Ausschalten der Repeatfunktion für die Tasten. Außerdem Fehlermodus: Tritt ein Programmfehler auf, so wird die Zeile automatisch(!) gelistet und der Cursor an der fehlerhaften Stelle positioniert. • **DUMP** listet alle normalen Variablen mit Inhalten. • **MATRIX** listet alle Feldvariablen (Arrays) mit Inhalten. • **LETTER (OFF)** Umschaltung Grafik-/Großkleinschreibmodus. • **FAST (OFF)** schnelle Bildschirmausgabe bei PRINT und LIST. • **"."** Symbol für „zuletzt bearbeitete Zeilennummer" (z.B.: 100 GETA$:IFA$ = ""THEN.) • **MEM** gibt Speicherplatzbelegung aus: MAIN MEMORY, PROGRAM, VARIABLES, ARRAYS, STRINGS, REK, FREE. • **HIMEM** Speicherplatzabgrenzung für Maschinenprogramme. • **GO** Einsprung in Monitor TIM bzw. EXMON. • **SPACE (OFF)** formatiertes Listen. • **STOP ON/OFF** Die Stoptaste bleibt auch beim Ablauf von Maschinenprogrammen aktiv. • **HELP** listet alle EXBASIC LEVEL II Befehle. • **HELP•** listet die Befehle des normalen Basic. • **BASIC** Ausschalten von EXBASIC LEVEL II. • **MERGE** Zusammenfügen von Basicprogrammen (auch Ineinander) von Kassette. • **MERGE•** wie MERGE, aber von Floppy.

Grafikbefehle:
PRINT AT Druck an spezifizierte Bildschirmstelle. • **HPLOT** horizontales Plotten. • **VPLOT** vertikales Plotten. • **SET** Grafikpunkt setzen. • **RESET** Grafikpunkt löschen. • **POINT** Abfrage, ob Grafikpunkt gesetzt oder nicht.

Mathematische Funktionen:
MAX sucht das Maximum aus einer Variablenliste. • **MIN** gibt das Minimum einer Variablenliste aus. • **FRAC(X)** Nachkommateil einer Zahl. • **ROUND(X)** rundet die Zahl X auf eine ganze Zahl. • **ROUND(X,Y)** rundet die Zahl X auf Y Nachkommastellen. • **ODD(X)** Abfrage ob X gerade oder ungerade ist. • **RND(X)** erzeugt Zufallszahlen ohne Nachkommateil zwischen 1 und X. • **HEX$(X)** wandelt die Zahl X in den entsprechenden hexadezimalen String um. • **DEC(string)** wandelt umgekehrt Hexstring in die Dezimalzahl.

LEVEL II Basic:
ELSE ermöglicht strukturierte IF..THEN..ELSE-Anweisungen, THEN kann bei Eindeutigkeit entfallen. • **RESTORE** mit Zeilennummernangabe für gezielten DATA-Zugriff. • **ON..RESTORE** gesteuertes RESTORE, ähnlich ON..GOTO. • **PRINT USING** Ausdruckformatierung für Bildschirm oder Drucker (z.B. für Komma unter Komma Druck von Zahlen). • **REK** Erweiterung der Unterprogrammebenen für rekursives Programmieren. • **DISPOSE NEXT** Stackrückstellung für FOR-NEXT-Schleifen („Notausgang" aus einer Schleife ohne NEXT). • **DISPOSE RETURN** Wechsel auf eine höhere Unterprogrammebene ohne RETURN-Sprung. • **DISPOSE CLR** Aussprung aus allen Schleifen und Unterprogrammen, unabhängig von Schachtelung und Tiefe. • **INPUTLINE** ermöglicht die Eingabe aller Zeichen, also auch Komma, Anführungszeichen, Steuerungszeichen, Doppelpunkt ohne Einschränkungen, bei leerer Eingabe wird das Programm nicht abgebrochen, sondern die Eingabevariable behält ihren vorherigen Inhalt. • **INPUTFORM** Eingabe von Standardzeichen (keine Cursorsteuerung oder RVS, aber Komma, Anführungszeichen etc.), sonst wie INPUTLINE, aber es kann eine bestimmte Eingabelänge spezifiziert werden und anderes. • **DEF USR=** definiert USR-Vektor. • **DEF CALL=** definiert CALL-Vektor. • **CALL** Aufruf von Maschinenprogrammen mit Parameterübergabe. • **DOKE** Doppelbyte-POKE bis 65535. • **DEEK** Doppelbyte-PEEK. • **VARPTR** gibt die RAM-Adresse an, wo eine bestimmte Variable abgespeichert ist. • **SPACE** Druck auf definiertes Bildschirmfeld. • **STRING$** Beispiel: A$=STRING$ (5,"-") ist gleichbedeutend mit A$="-----". • **INSTR** untersucht ob ein String in einem anderen enthalten ist. • **EVAL** ähnlich VAL, aber Bsp.: VAL ("12•4") ist 12, während EVAL ("12•4") den Wert 48 ergibt. EVAL verarbeitet alle Funktionen. • **EXEC** führt Basic-Befehle im String aus. • **SWAP** vertauscht zwei Variablen ohne Zuhilfenahme einer dritten. • **SEC** Programmpause für X Sekunden. • **BEEP** Tonerzeugung an CB2. Tonhöhe und -dauer können mit angegeben werden. • **ON ERROR GOTO** ermöglicht die selbständige Fortsetzung eines Programmes nach Auftreten eines Programmfehlers. Zur Fehlerbehandlung stehen die Variablen EC (error code) und EL (error line) zur Verfügung. • **RESUME** Abschluß einer Fehlerbehandlungsroutine, die mit ON ERROR GOTO angesprungen wird, mit verschiedenen Möglichkeiten: RESUME; RESUME NEXT, RESUME Zeilennummer. • **"'"** Ersatzsymbol für den Befehl REM. • **HARDCOPY** kopiert den Bildschirm auf einen Drucker.

FAST TAPE (CBM 2/3/4; VC 20/64)
Für die genannten Serien stehen neben den normalen Kassettenrekorderbefehlen neue zur Verfügung, die mit 5facher Geschwindigkeit arbeiten. Programmnamen dürfen bis zu 30 Zeichen (normal 16) lang sein. **MOD** erzeugt ein Programm, das von jedem anderen Computer auch ohne EXBASIC LEVEL II fünfmal so schnell eingeladen werden kann und egal wie es eingelesen wird, automatisch startet.

DOS SUPPORT (CBM 2/3/4; VC 20/64)
Das DOS SUPPORT bietet komfortable Unterstützung beim Arbeiten mit einer Floppystation durch die Verwendung von Kurzbefehlen. Mit EXEC sind alle DOS SUPPORT Funktionen auch programmierbar.

SCREEN SUPPORT (CBM 8):
Bildschirmsonderbefehle. **SCREEN** definiert Bildschirmfenster. • **DELLINE** löscht eine oder mehrere Bildschirmzeilen. • **INSTLINE** schafft Platz für eine oder mehrere Bildschirmzeilen. • **ENDLINE** löscht ab Cursorposition bis Zeilenende. • **BEGINLINE** löscht vom Zeilenanfang bis zur Cursorposition. • **SCREEN UP/DOWN** Bildschirmrollen nach oben und unten • **SCREEN •** macht den 80-Zeichen-Bildschirm des CBM 8000 kompatibel zum 40-Zeichen-Schirm der anderen Serien

EXMON (CBM 8):
Für die Serie 8000 enthält EXBASIC LEVEL II einen Maschinensprachemonitor, der mit GO aufgerufen wird und folgende Befehle bietet: **A** — Assemblieren • **C** — Berechnung eines relativen Offsets • **D** — Disassemblieren auf dem Bildschirm • **G** — Ausführung eines Assemblerprogrammes • **L** — Laden eines Programmes (Kassette oder Diskette) • **M** — Hexdump • **P** — Disassemblieren auf einem Drucker • **S** — Abspeichern eines Assemblerprogrammes

COLOR SUPPORT (VC 20/64)
mit speziellen Kommandos zur Farbgestaltung.

SOUND SUPPORT (VC 20/64)
Sonderbefehle für die Ton- und Soundprogrammierung

KEY DEFINER (VC 20/64)
erlaubt Tastendefinitionen

Der Preis? Er ist – gemessen an den vielfältigen Möglichkeiten, die **EXBASIC LEVEL II** bietet – so klein, daß wir ihn hier fast vergessen hätten: DM 392,– inkl. MwSt. (unverbindliche Verkaufsempfehlung).